

Introduction to the analysis package.

Yo Sato (Tohoku University)

Belle II Starter Kit workshop, KEK, January 2020



HELMHOLTZ RESEARCH FOR
GRAND CHALLENGES

DESY.



Istituto Nazionale di Fisica Nucleare
SEZIONE DI TORINO



Contents

An experiment's workflow

What is basf2?

Invoking basf2 and the equivalent of “Hello World”

Modules, paths, the DataStore and how to steer them all

A word about file types and why you should use the analysis package

Now let's step through an example

Candidate/particle based analysis

Physics quantities and the VariableManager

How to get data out

Nomenclature

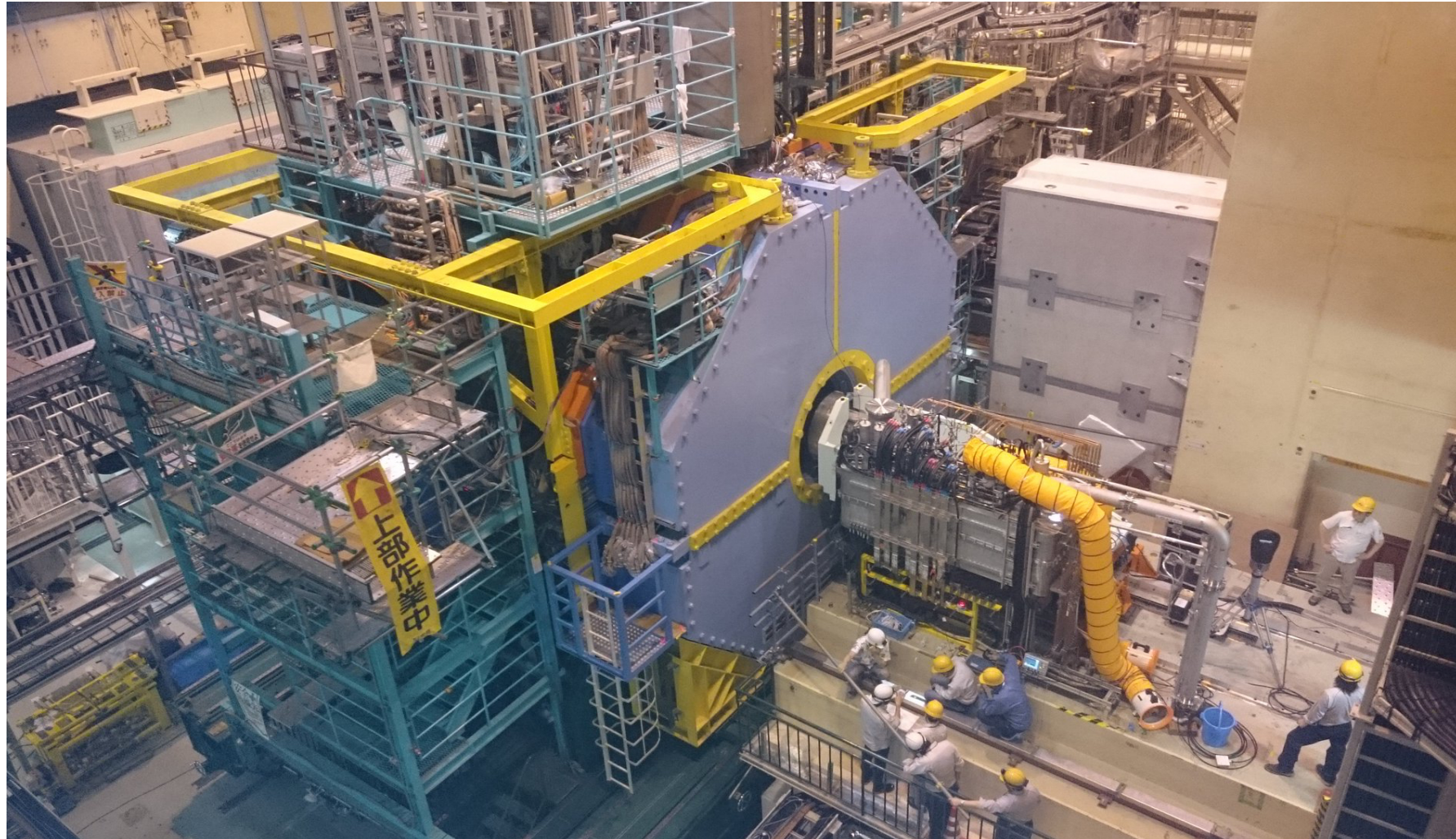
One final thing about Belle vs. Belle II



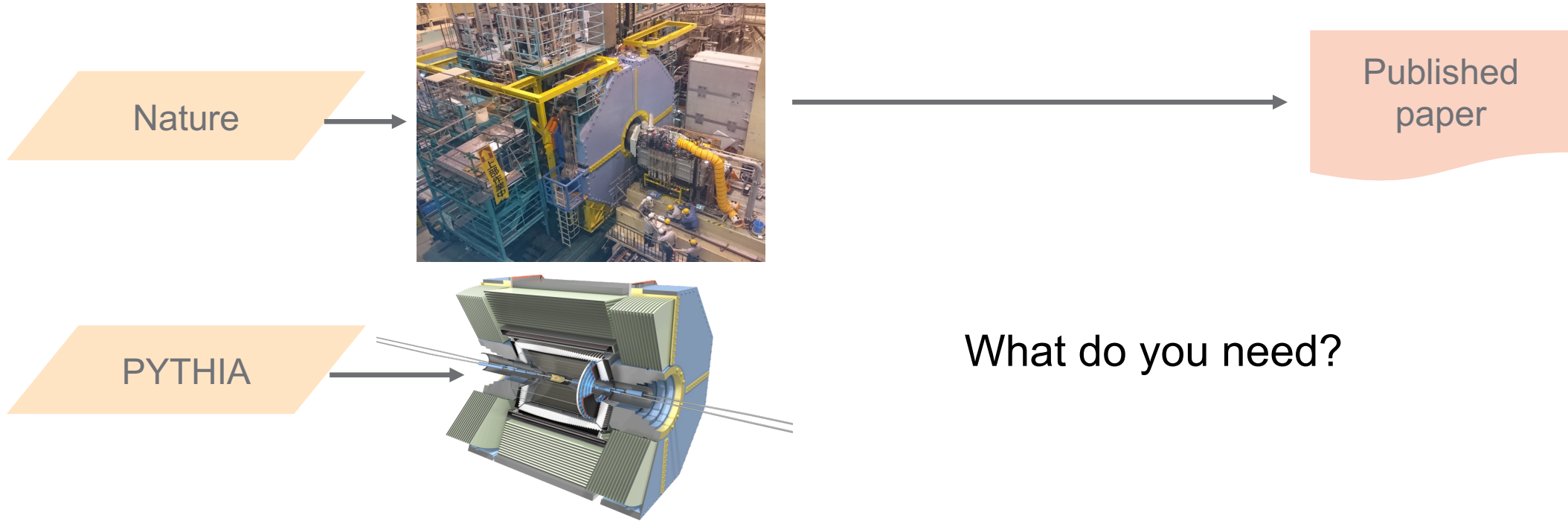
An experiment's workflow

The big picture

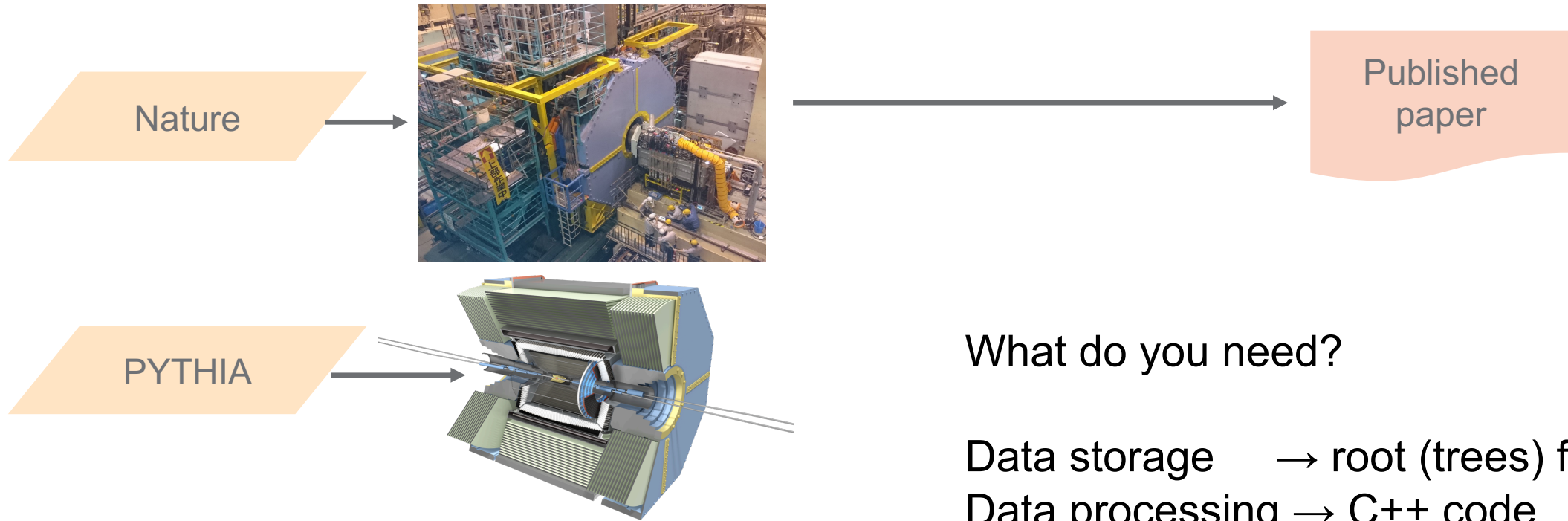
Tsukuba Hall



The big picture



The big picture



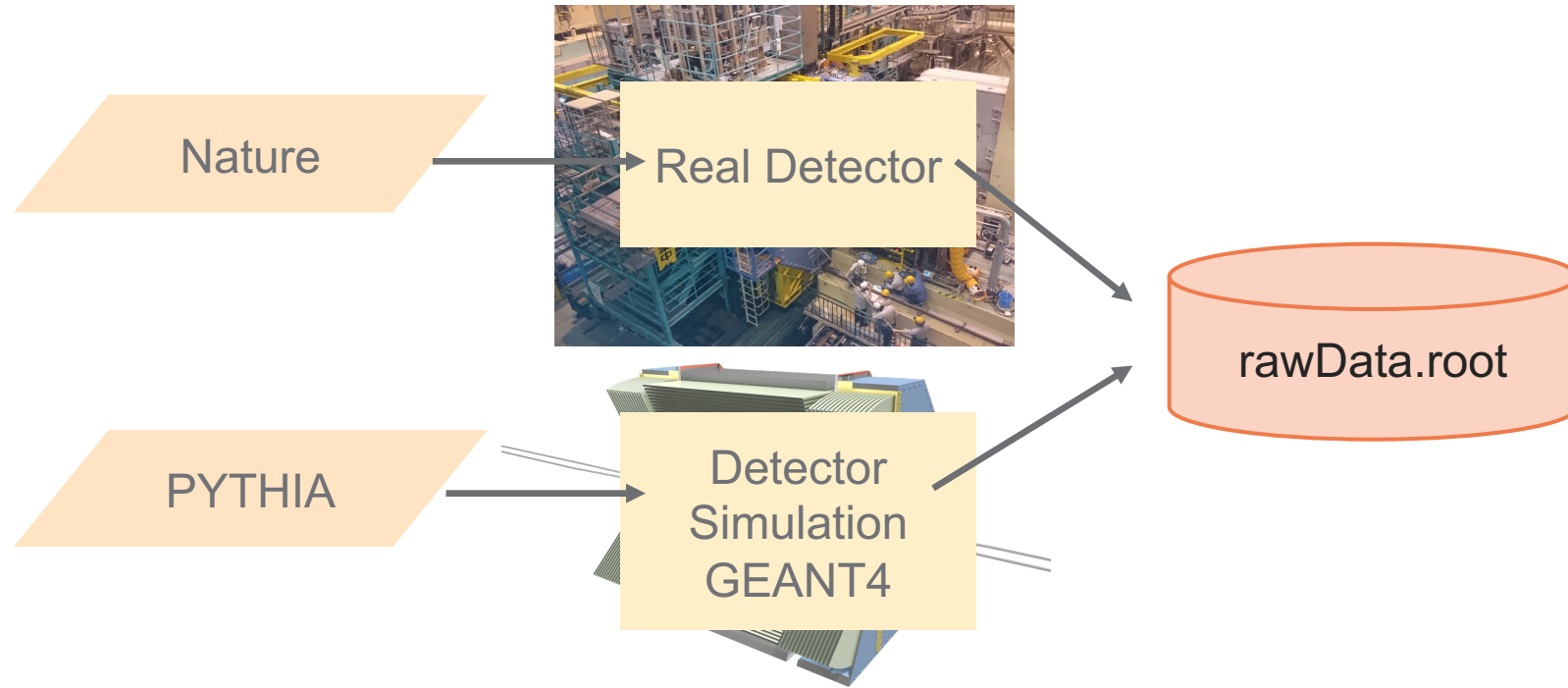
What do you need?

Data storage → root (trees) files

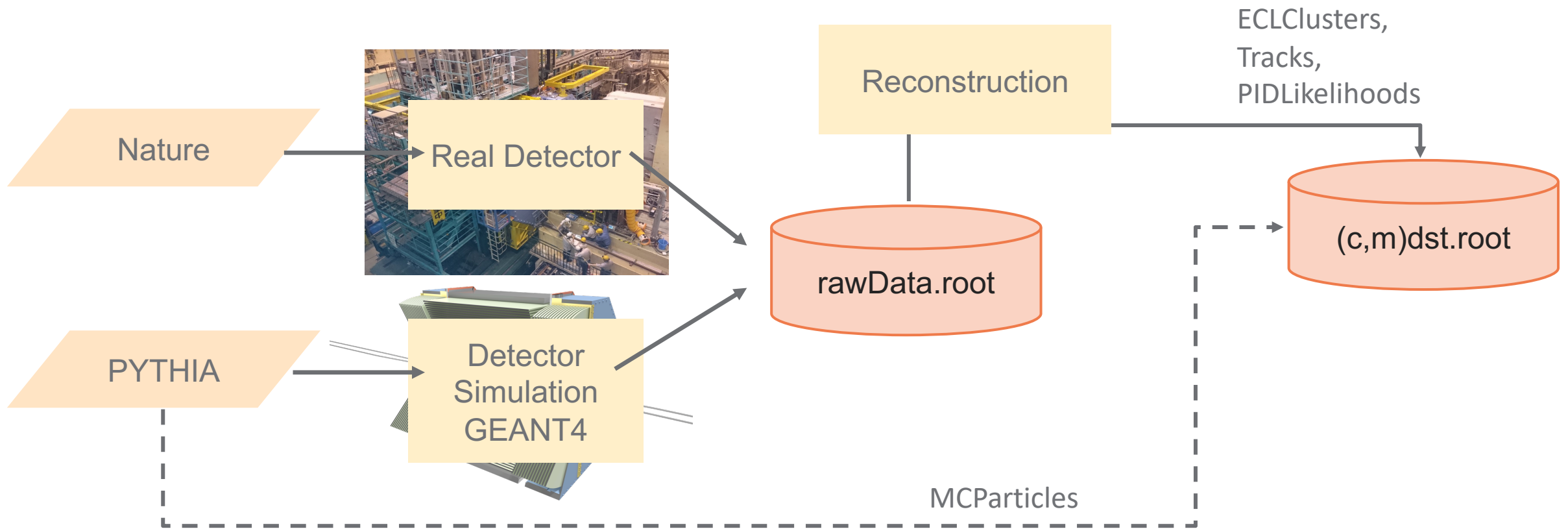
Data processing → C++ code

Scripting → Python

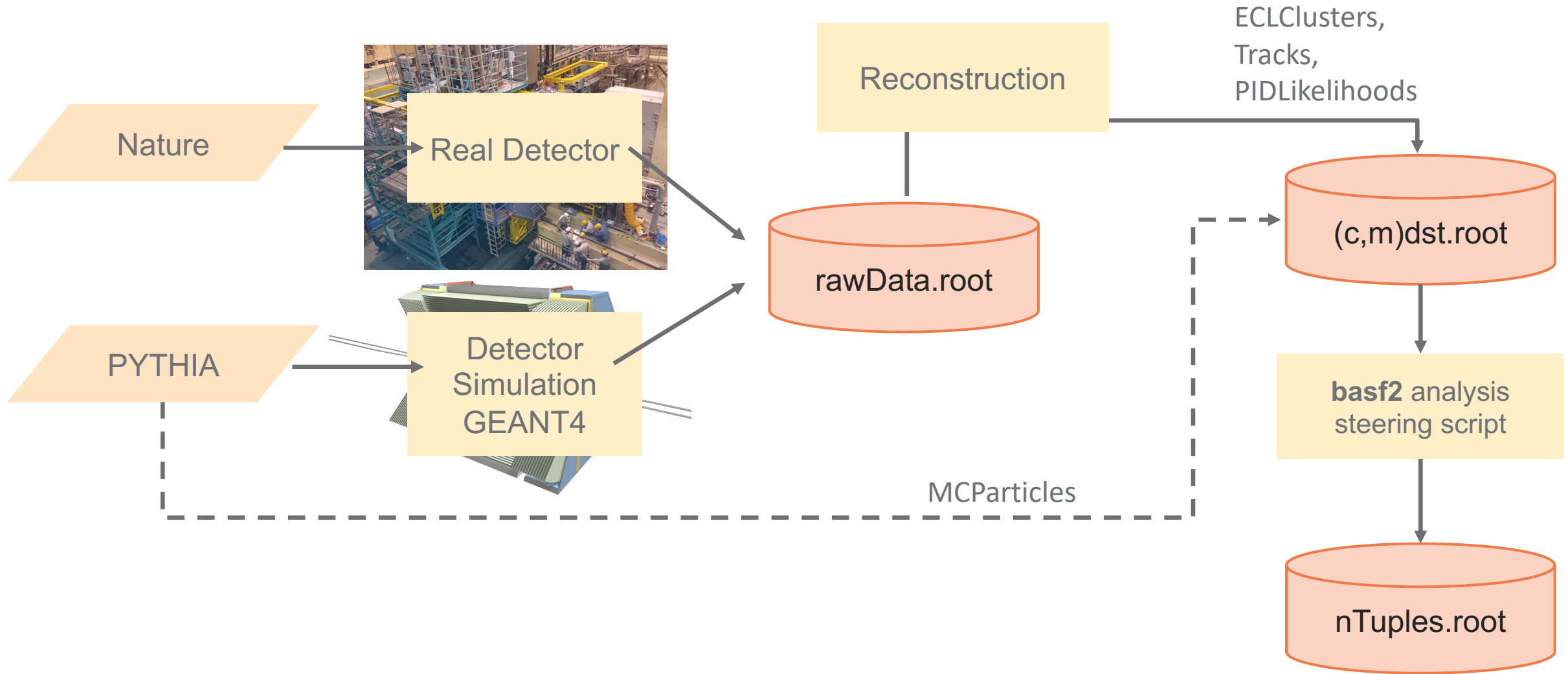
The big picture



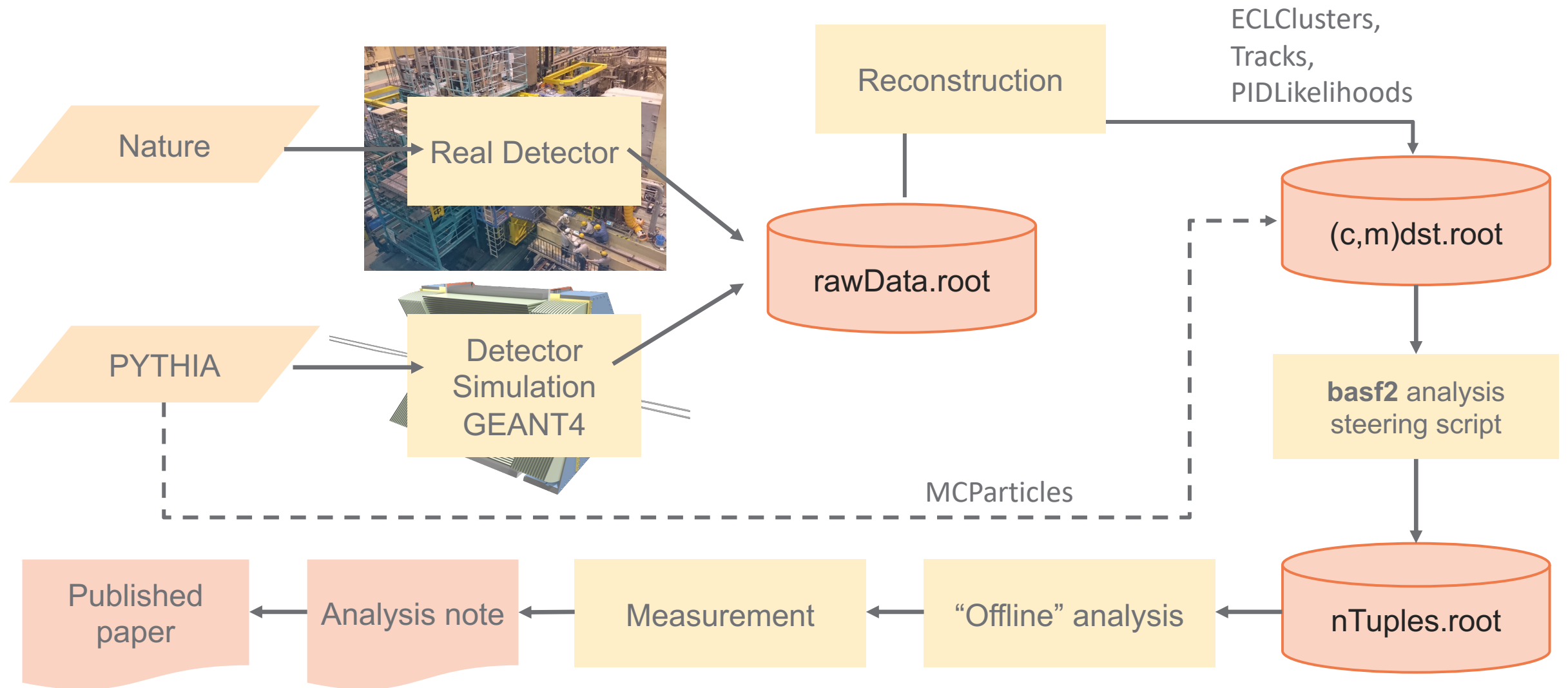
The big picture



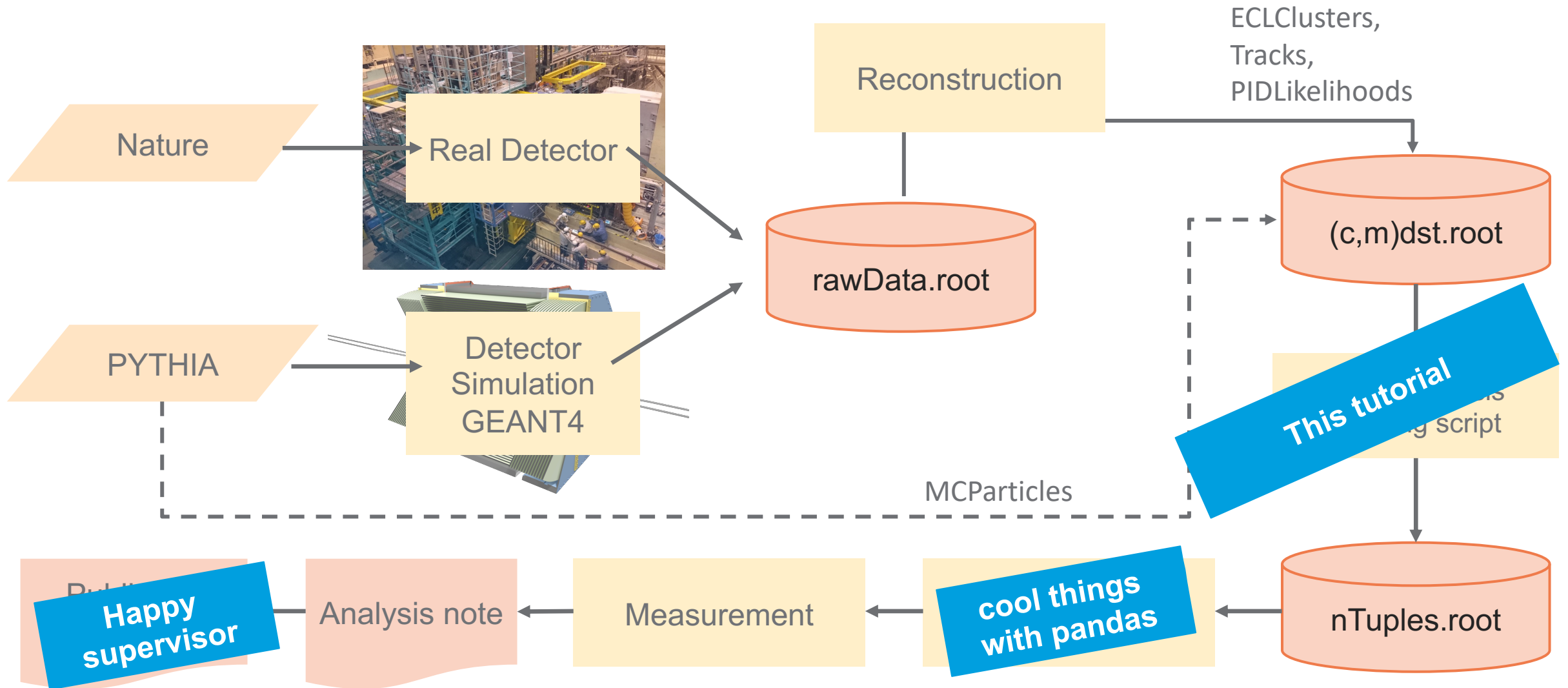
The big picture



The big picture



The big picture



What is basf2?

The code

basf2 is C++14 "**under the hood**"

- Packages contain C++ **modules** to manipulate data.
- In analysis: we have code to build **particles** from primitive objects (like tracks and calorimeter clusters).
- We also calculate physics quantities, and apply cuts.

Python 3.6 code for **steering**

- Load and configure C++ modules
 - analysis modules and modules from other packages
- Also python does *some* high-level analysis tasks.
- You will write a fair bit of python during the workshop.



The code

basf2 is C++14 "under the hood"

- Packages contain C++ **modules** to manipulate data.
- In analysis: we have **objects** from primitive objects (like tracks and calorimeters).
- We also calculate **observables** quantities, and apply cuts.

Heavy lifting

Python 3.6 code for **steering**

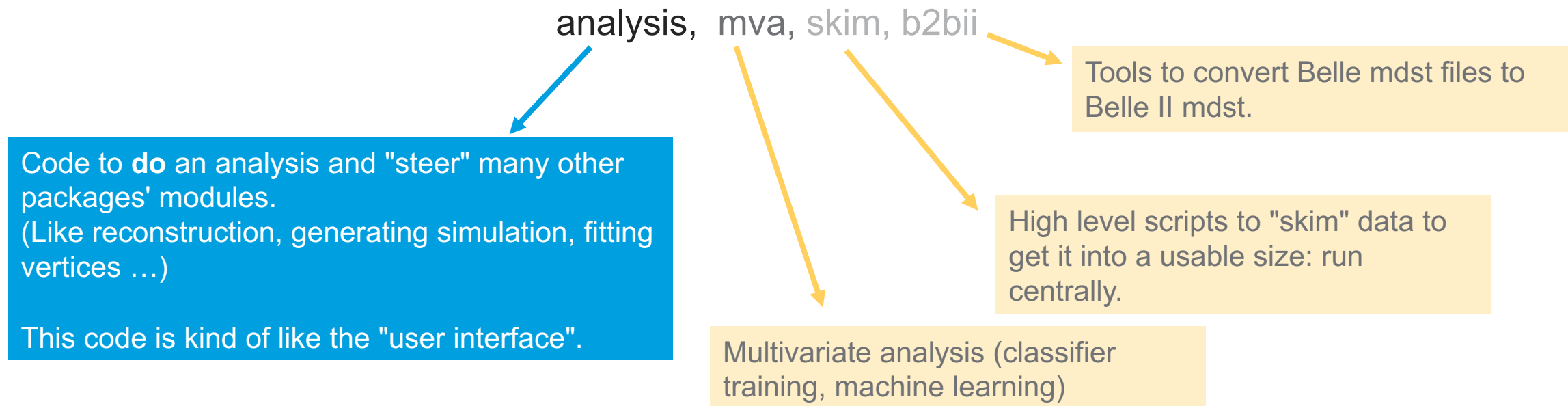
- Load and configure C++ modules
 - analysis modules and meta-packages
- Also python does **bookkeeping** tasks.
- You will write a **script** for python during the workshop.

Readable scripts



What is the analysis package?

- Our software is organised into "packages".
 - <https://stash.desy.de/projects/B2/repos/software/browse>
 - There are packages for subdetectors, tracking, simulation...
 - As a student/postdoc/collaborator you might work on some of them.
- BUT! When you want to do a physics measurement. You really only care about:



I'm stuck, where do I go for help?

Probably the most important slide

questions

for anything really, not just analysis package

<https://questions.belle2.org>

documentation

there is fairly good documentation

<https://software.belle2.org>

examples

so, for us <packagename> = analysis

`$BELLE2_RELEASE_DIR/<packagename>/examples`

the code

<https://stash.desy.de/projects/B2/repos/software/browse>

`$BELLE2_RELEASE_DIR`

I'm stuck, where do I go for help?

Probably the most important slide

questions

for anything really, not just analysis package

<https://questions.belle2.org>

documentation

there is fairly good documentation

<https://software.belle2.org>

examples

so, for us <packagename> = analysis



Go here first!

the code

[\\$BELLE2_RELEASE_DIR/<packagename>/examples](https://stash.desy.de/projects/B2/repos/software/browse)

Invoking basf2 and the equivalent of “Hello World”

How to invoke basf2

Belle analysis software framework 2

- Let's setup basf2 !
- But, first of all, you should know which release can be and should be used.

```
$ . /cvmfs/belle.cern.ch/tools/b2setup
$ b2setup --help # You can see available releases of basf2
...
  The following releases are available:
...
  release-04-01-00
  release-04-01-01
  light-1912-icarus

$ b2help-releases # or you can ask the recommended release
release-04-01-01
```

How to invoke basf2

Belle analysis software framework 2

Q. Which release should I use?

A. I really recommend you to use the latest `release-AA-BB-CC` or `light-YYMM-CODENAME`.

`release-AA-BB-CC`

A full release, such as `release-04-01-01` and `release-03-02-04`.

If you execute ``b2help-releases``, it tells the latest full release.

`light-YYMM-CODENAME`

A light release, such as `light-1912-icarus` and `light-1911-heracles`.

Made from only a few packages, analysis, mdst, skim, b2bii (from light-1912) etc..

They are suitable for doing analysis!

How to invoke basf2

Belle analysis software framework 2

- Hopefully you've seen this already.
- If not you will see it many times in this workshop.
- At the command line:

```
$ . /cvmfs/belle.cern.ch/tools/b2setup  
$ b2setup release-04-01-01  
$ basf2 --info
```

How to invoke basf2

Belle analysis software framework 2

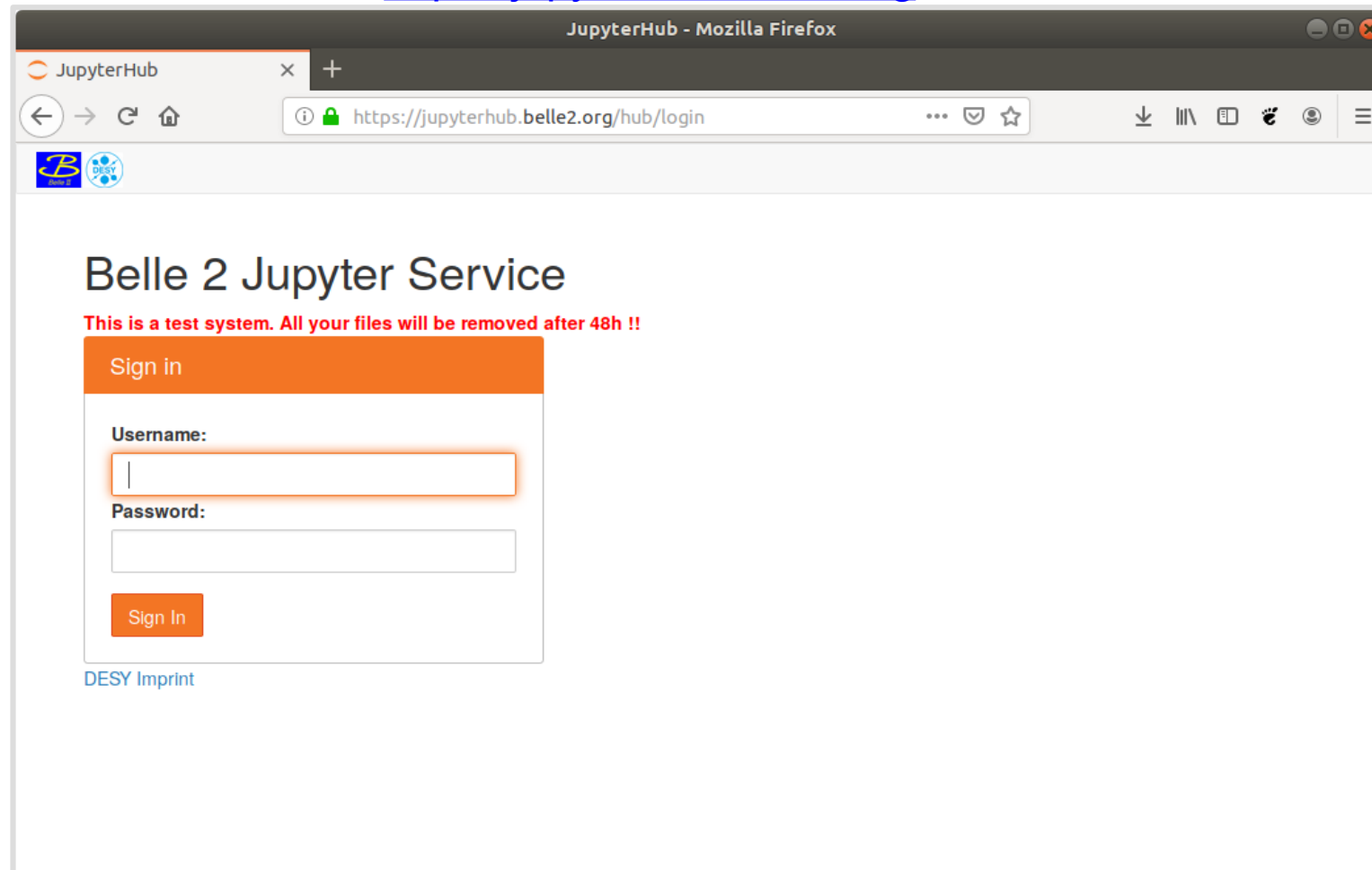
- Then Let's setup basf2 for the release!
- ``. /cvmfs/belle.cern.ch/tools/b2setup`` needs only once per session.

```
$  
$  
$ . /cvmfs/belle.cern.ch/tools/b2setup release-04-01-01  
$ basf2 --info
```

Today we'll use jupyter

You will cover command line usage later in the workshop

<https://jupyterhub.belle2.org>



Start a server, open the first exercise...

<https://jupyterhub.belle2.org>

B2T_Basics_1_GettingHelp (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (Belle2)

BASF2 Basics

If you run things at the command line (for example at KEKCC or NAF) you will need to set up your environment and whatever release.

```
source /cvmfs/belle.cern.ch/tools/b2setup
b2setup release-03-01-01
```

Actually if you already know what release you want then you can do it all in once step:

```
source /cvmfs/belle.cern.ch/tools/b2setup release-03-01-01
```

To list all available releases:

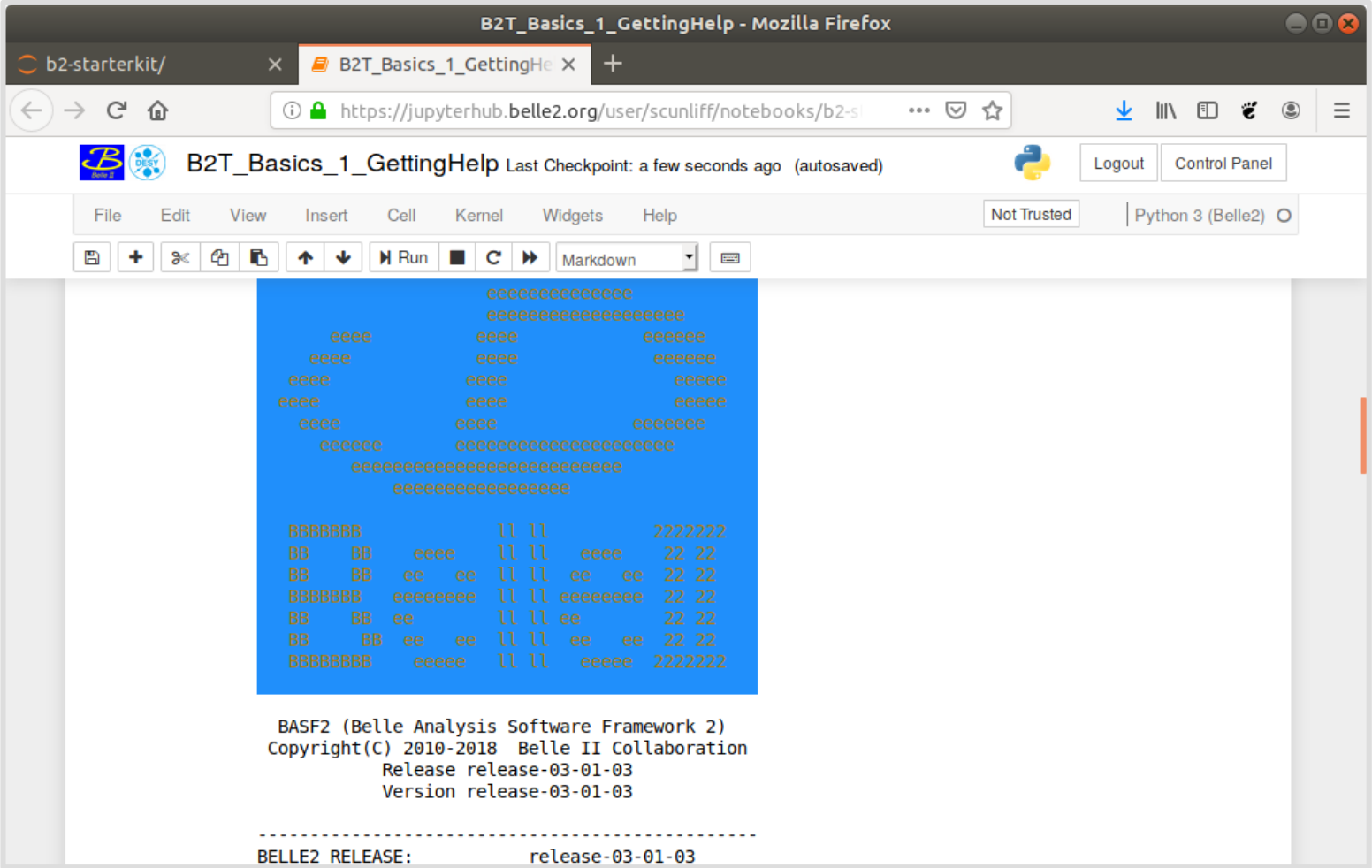
```
b2setup --help
```

You **don't** need to do that in this ipython/jupyter environment (we've set it all up for you). But please remember this, you'll need it later.

```
In [ ]: !basf2 --info
```


basf2 --info

<https://jupyterhub.belle2.org>



basf2 --info

Also works on a terminal to KEKCC or NAF

Source the logon script,
pick a release

Execute this

```
Terminal
File Edit View Search Terminal Help
> source /cvmfs/belle.cern.ch/tools/b2setup release-03-01-03
Belle II software tools set up at: /cvmfs/belle.cern.ch/tools
Environment setup for release: release-03-01-03
Central release directory   : /cvmfs/belle.cern.ch/ubuntu1804/releases/release-03-01-03

> basf2 --info

          eeeeeee
        eeeeeeeeeeeeeee
      eeeeeee   eeeeeeeeeeeeeee
     eeeeeee   eeee   eeeeeee
    eeee       eeee   eeee
   eeee       eeee   eeee
  eeee       eeee   eeee
 eeee       eeeeeeeeeeeeeee
          eeeeeeeeeee

          eeeeeeeeeee
        eeeeeeeeeeeeeee
      eeee   eeee   eeeee
     eeee   eeee   eeeee
    eeee   eeee   eeeee
   eeee   eeee   eeeee
  eeee   eeee   eeeee
 eeeee   eeeeeeeeeeeeeee
          eeeeeeeeeeeeeee
          eeeeeeeeeeeeeee

BBBBBBB   ll ll   222222
BB  BB   eeee   ll ll   eeee   22 22
BB  BB   ee  ee  ll ll   ee  ee  22 22
BBBBBBB   eeeeeee ll ll   eeeeeee 22 22
BB  BB   ee  ee  ll ll   ee  ee  22 22
BB  BB   ee  ee  ll ll   ee  ee  22 22
BBBBBBB   eeee   ll ll   eeee   222222

BASF2 (Belle Analysis Software Framework 2)
Copyright(C) 2010-2018 Belle II Collaboration
Release release-03-01-03
Version release-03-01-03

-----
BELLE2_RELEASE:      release-03-01-03
BELLE2_RELEASE_DIR:  /cvmfs/belle.cern.ch/ubuntu1804/releases/release-03-01-03
```

basf2 --info

Also works on a terminal to KEKCC or NAF

Useful information

```
Terminal
File Edit View Search Terminal Help
      eeeee      eeee      eeee
      eeee      eeeeeeeeeeeeeee
                        eeeeeeeeeeeeeee
                        eeeeeeeeeeeeeee
      eeee      eeee      eeeee
      eeee      eeee      eeeee
      eeee      eeee      eeeee
      eeee      eeee      eeeee
      eeeee     eeeeeeeeeeeeeee
      eeeeeeeeeeeeeeeeeee
      eeeeeeeeeeeeeee

BBBBBBB      ll ll      2222222
BB  BB      eeee      ll ll      eeee      22 22
BB  BB      ee  ee      ll ll      ee  ee      22 22
BBBBBBB      eeeeeee      ll ll      eeeeeee      22 22
BB  BB      ee  ee      ll ll      ee  ee      22 22
BB  BB      ee  ee      ll ll      ee  ee      22 22
BBBBBBB      eeeee      ll ll      eeeee      2222222

BASF2 (Belle Analysis Software Framework 2)
Copyright(C) 2010-2018 Belle II Collaboration
Release release-03-01-03
Version release-03-01-03

-----
BELLE2_RELEASE:      release-03-01-03
BELLE2_RELEASE_DIR:  /cvmfs/belle.cern.ch/ubuntu1804/releases/release-03-01-03
BELLE2_LOCAL_DIR:
BELLE2_SUBDIR:      Linux_x86_64/opt
BELLE2_EXTERNALS_VERSION: v01-07-01
BELLE2_ARCH:      Linux_x86_64
Default global tags:  release-03-01-00_rev3
Kernel version:      4.15.0-48-generic
Python version:      3.6.6
ROOT version:      6.14/06

basf2 module directories:
/home/scunliff
/cvmfs/belle.cern.ch/ubuntu1804/releases/release-03-01-03/modules/Linux_x86_64/opt
-----
>
```

Modules, paths, the DataStore and how to steer them all

Modules, paths, the DataStore

What do we need to process the data?

Kuhr, Pulvermacher, Ritter, Hauth, Braun
Comput. Softw. Big Sci. 3 (2019) no.1

- 1) A set of classes (modules) that process the data
→ **BASF2 module**

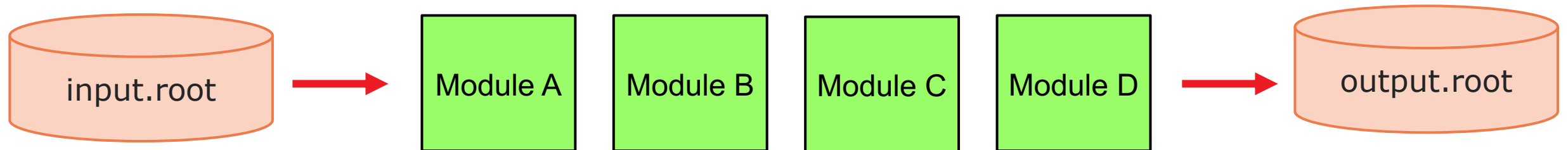
A module is written in C++ or Python and derived from a `Module` base class that defines the following interface methods:

- `initialize()`: called before the processing of events to initialize the module.
- `beginRun()`: called each time before a sequence of events of a new run is processed, e.g., to initialize run-dependent data structures like monitoring histograms.
- `event()`: called for each processed event.
- `endRun()`: called each time after a sequence of events of the same run is processed, e.g., to collect run-summary information.
- `terminate()`: called after the processing of all events.

Modules, paths, the DataStore

What do we need to process the data?

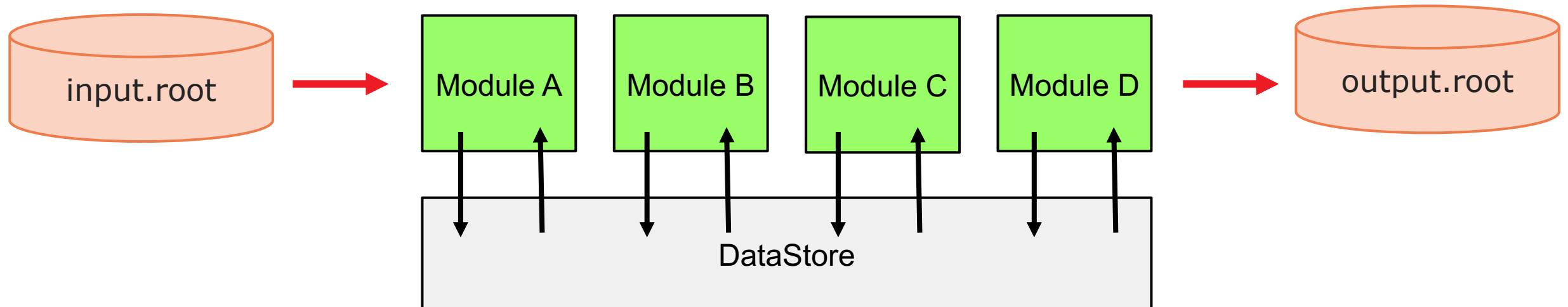
- 1) A set of classes (modules) that process the data
→ **BASF2 module**



Modules, paths, the DataStore

What do we need to process the data?

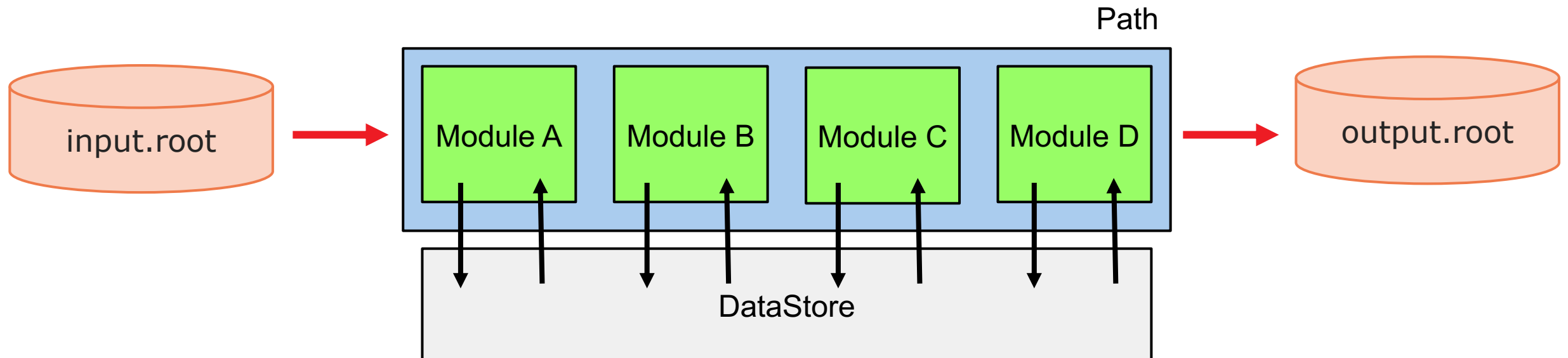
- 1) A set of classes (modules) that process the data
→ **BASF2 module**
- 2) A set of classes (dataobjects) that hold the data and allow module to pass thing one to the other
→ **BASF2 dataStore**



Modules, paths, the DataStore

What do we need to process the data?

- 1) A set of classes (modules) that process the data
→ **BASF2 module**
- 2) A set of classes (dataobjects) that hold the data and allow module to pass thing one to the other
→ **BASF2 dataStore**



- 3) An order in which the modules must be executed
→ **BASF2 path**

What does a steering file look like?

How to I implement all this?

A steering file is a python script that runs

- the modules that you need
- in the order you need
- on the data you need

A word about file types and why you should use the analysis package

File types

That basf2 can read and/or create

- A **dst** contains basf2 objects which will populate a **DataStore**.
 - **data summary table**
 - Basically: a special ROOT file.
- The data for physics analysis are "**mdst**"
 - **mini data summary table**.
 - Same structure of a **dst**, **but with much less information**
 - Input to your analysis package scripts
- The calibration & performance are "**cdst**"
 - **calibration data summary table**.
 - mdst + digits
- At the end of your analysis chain you will write out a "normal" root file containing a TTree, TNtuple, or histograms



A relevant question

<https://questions.belle2.org/question/219>

Objects allowed in an *mdst*:

<https://goo.gl/AB15Ud>

Why use the analysis package?

Can I read the mdst by myself?

mdst are basically root trees containing lists of:

- Track
- TrackFitResult
- V0
- PIDLikelihood
- ECLCluster
- KLMCluster
- Kllid
- TRGSummary
- SoftwareTriggerResult
- (MCParticle)
- ...

The analysis package has modules to convert these into more friendly quantities like

- Particle
- ParticleList
- EventShapeContainer
- TagVertex
- ...

Why use the analysis package?

Can I read the mdst by myself?

Can I open the mdst with my own, custom-made macro and run the analysis?

Why use the analysis package?

Can I read the mdst by myself?

Can I open the mdst with my own, custom-made macro and run the analysis?

NO

The mdst contains also the relations between the objects stored in it, which are not trivially handled by a stand-alone root macro. **Use always basf2-based code.**

Why use the analysis package?

Can I read the mdst by myself?

Can I open the mdst with my own, custom-made macro and run the analysis?

NO

The mdst contains also the relations between the objects stored in it, which are not trivially handled by a stand-alone root macro. **Use always basf2-based code.**

Should I write my own module that loops over reconstructed objects like the ECLClusters and do the analysis (i.e. Belle-style)?

Why use the analysis package?

Can I read the mdst by myself?

Can I open the mdst with my own, custom-made macro and run the analysis?

NO

The mdst contains also the relations between the objects stored in it, which are not trivially handled by a stand-alone root macro. **Use always basf2-based code.**

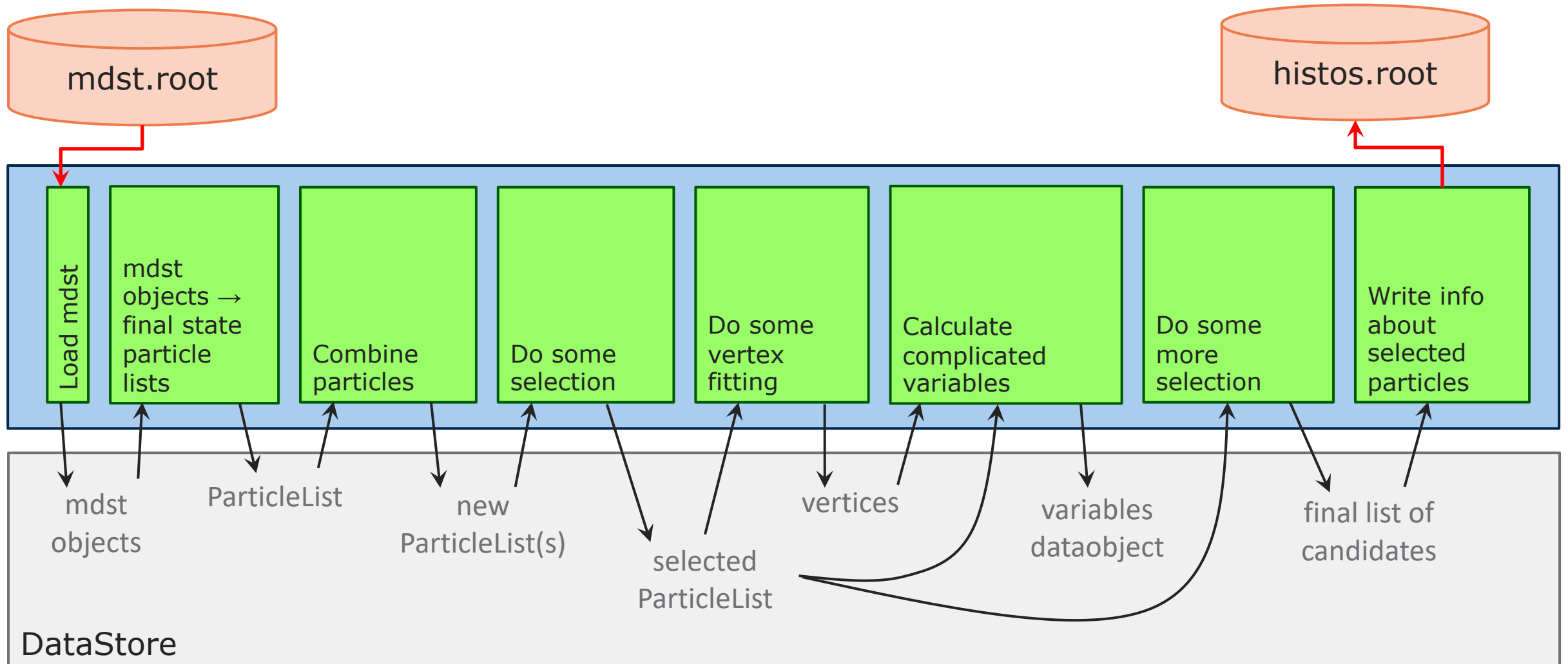
Should I write my own module that loops over reconstructed objects like the ECLClusters and do the analysis (i.e. Belle-style)?

NO

The relation between analysis object (particles) and the reconstructed objects is not always trivial.
One particle may have many trackFitResults
The ECLClusters are not photons.
Use the modules provided by a detector expert

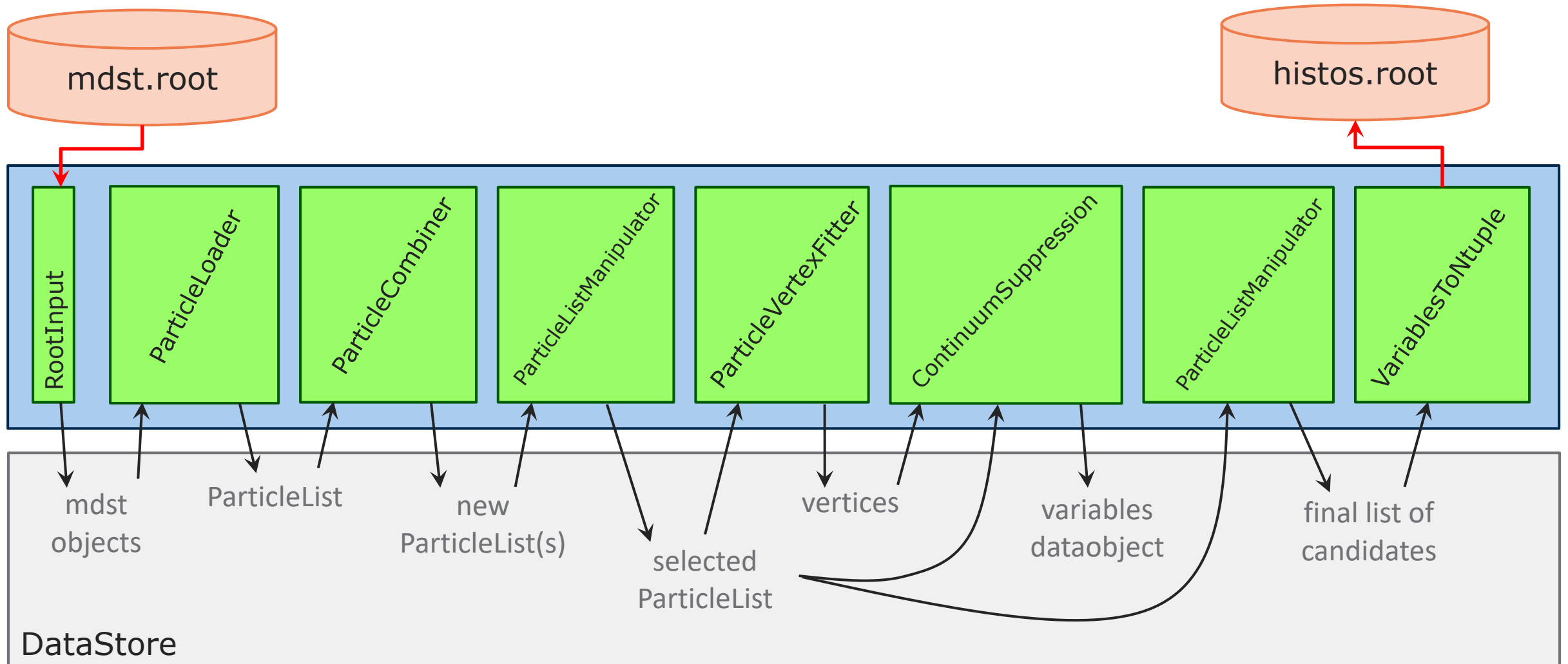
**Now let's step through an
example**

A typical path for an analysis job



A typical path for an analysis job

Now with the real names for the modules



How to make a path and load a module

- First of all, you have make your path.

```
import basf2

mypath = basf2.Path()
# mypath basf2.create_path() # Both are OK!
```

How to make a path and load a module

- Call a function to load a module and add it to your path.
- That is all you need to do!

```
from modularAnalysis import fillParticleList  
  
fillParticleList('pi+:highMom', 'p > 1', path=mypath)
```

Load a module to fill a list of particle, ParticleList.
We'll see details of ParticleList later.

- What is happening in the function?

The ParticleLoader

An analysis module which loads particles

```
fillParticleList('pi+:highMom', 'p > 1', path=mypath)
```



```
pload = register_module('ParticleLoader')
pload.param('decayStringsWithCuts', [('pi+:highMom', 'p > 1')])
mypath.add_module(pload)
```

- The function wraps these steps in a line.
- Of course, you can write them by hand.
 - **But quite quickly scripts become unreadable**

The ParticleLoader

An analysis module which loads particles

- The procedure is same no matter what module you want in path.
 - "Register" the module you want
 - Set "parameters" to the module
 - "Add" the module to you path

```
pload = register_module('ParticleLoader')
pload.param('decayStringsWithCuts', [('pi+:highMom', 'p > 1')])
mypath.add_module(pload)
```

How to find module documentation

In general

- At the command line:

```
$ basf2 -m  
$ basf2 -m ParticleLoader
```

Take a look at
B2T_Basics_1_GettingHelp.ipynb
on jupyterhub

Q: Can you find the **source code** for ParticleLoader?
What do you notice about the C++ class name? How does this compare to
the module name?

Wrapper functions

- Functions such as `fillParticleList` are found in `<package>/scripts/*.py`
- The script that you will probably use a lot is **`analysis/scripts/modularAnalysis.py`**

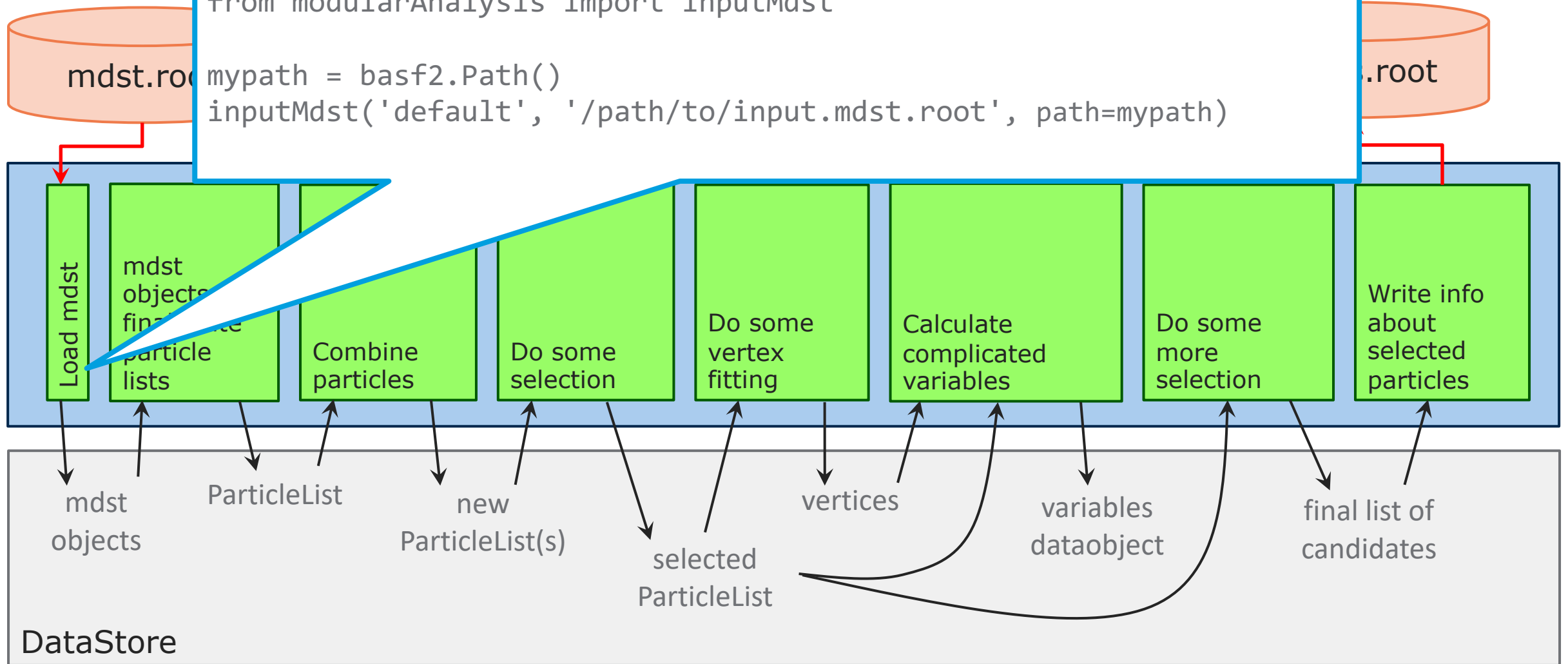
```
# analysis/scripts/modularAnalysis.py
from modularAnalysis import fillParticleList
fillParticleList('pi+:highMom', 'p > 1', path=mypath)
```

```
# analysis/scripts/vertex.py
from vertex import fitVertex
fitVertex('K*0:myKst', conf_level=0.0, path=mypath)
```

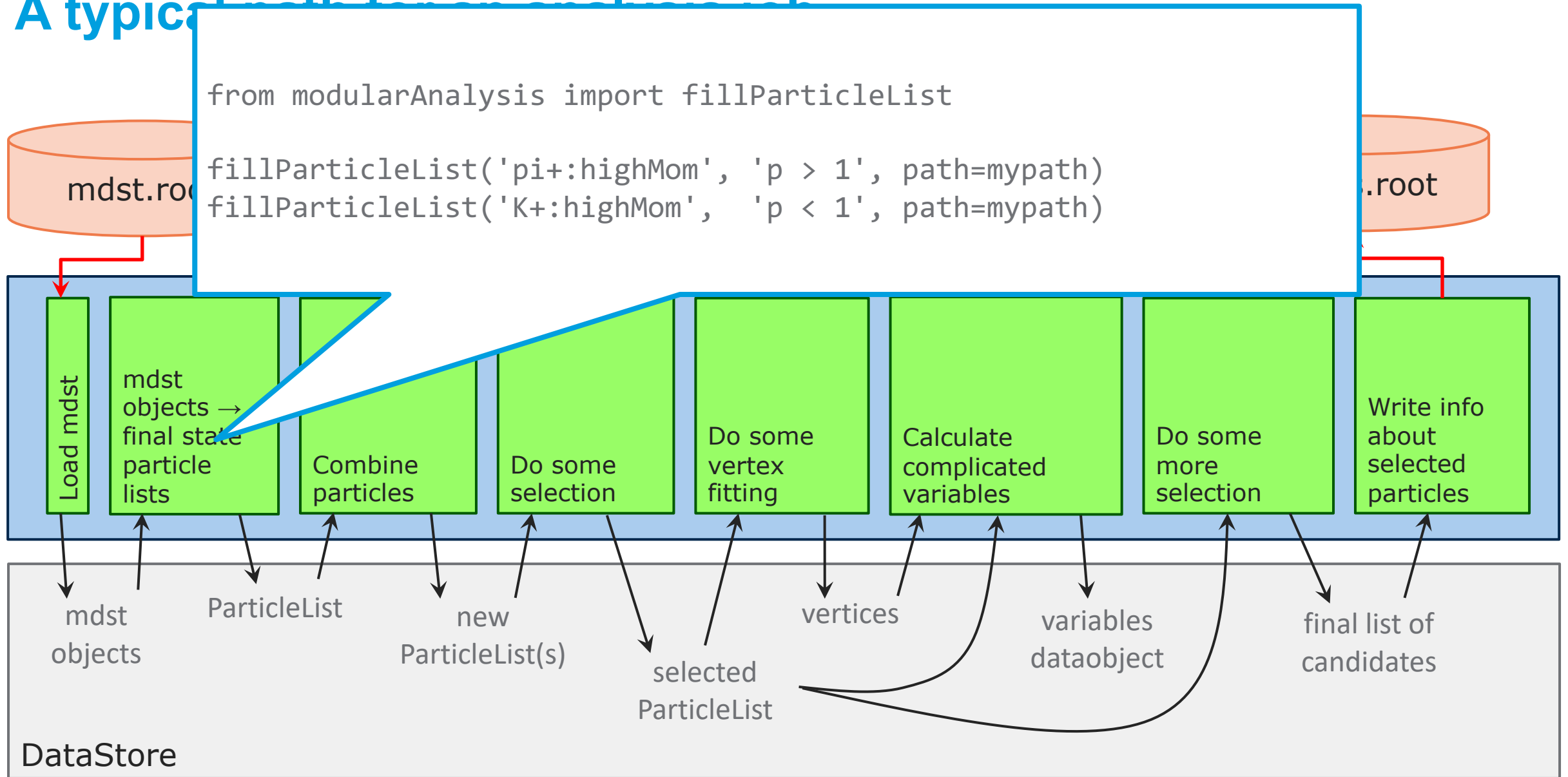
A typical analysis workflow

```
import basf2
from modularAnalysis import inputMdst

mdst.root mypath = basf2.Path()
inputMdst('default', '/path/to/input.mdst.root', path=mypath)
```

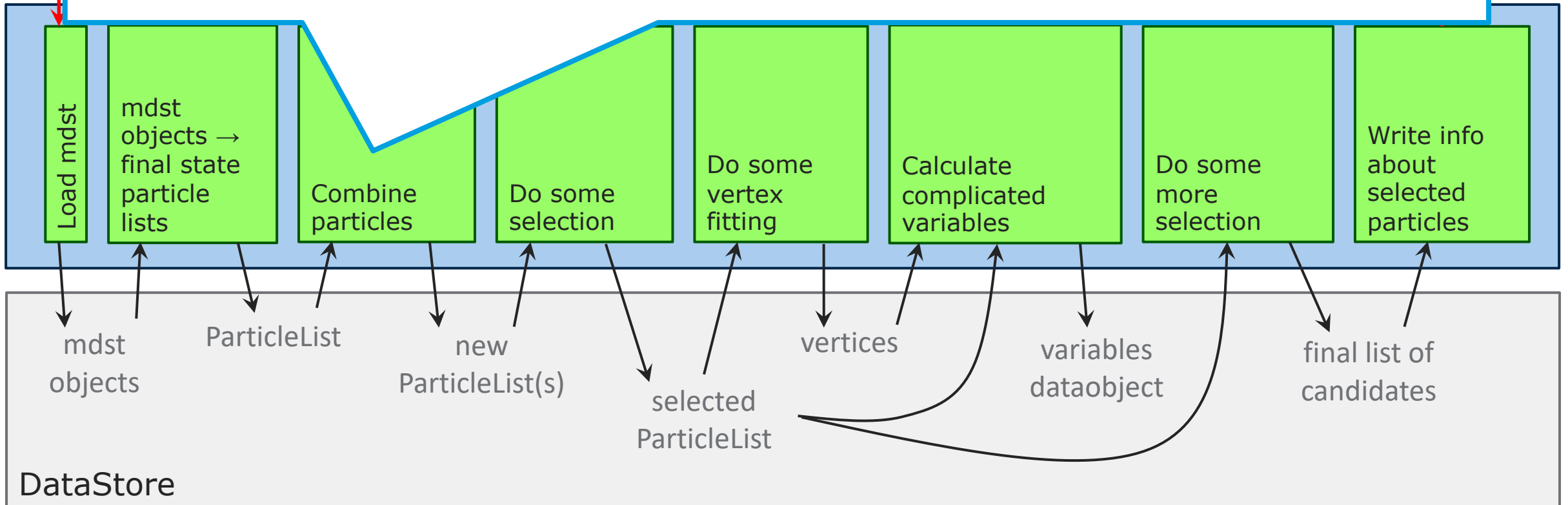


A typical event analysis pipeline



A

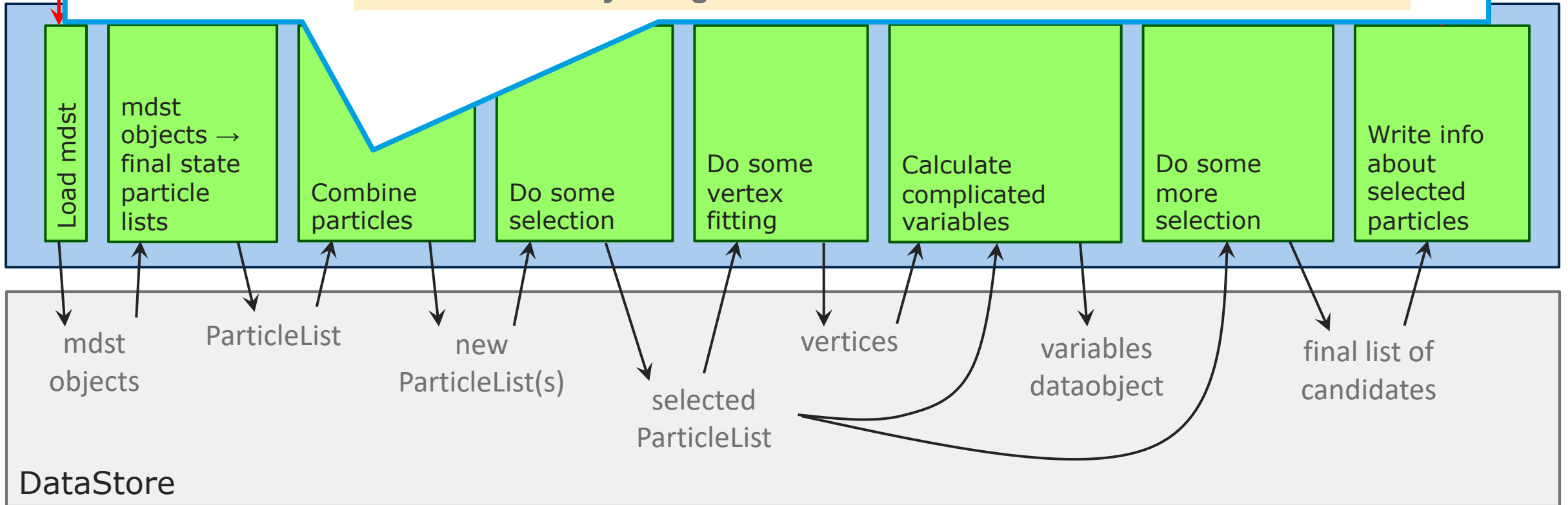
```
from modularAnalysis import reconstructDecay
reconstructDecay('K*0:myKst -> K+:highMom pi-:highMom', '0.6 < M < 1.0', path=mypath)
```



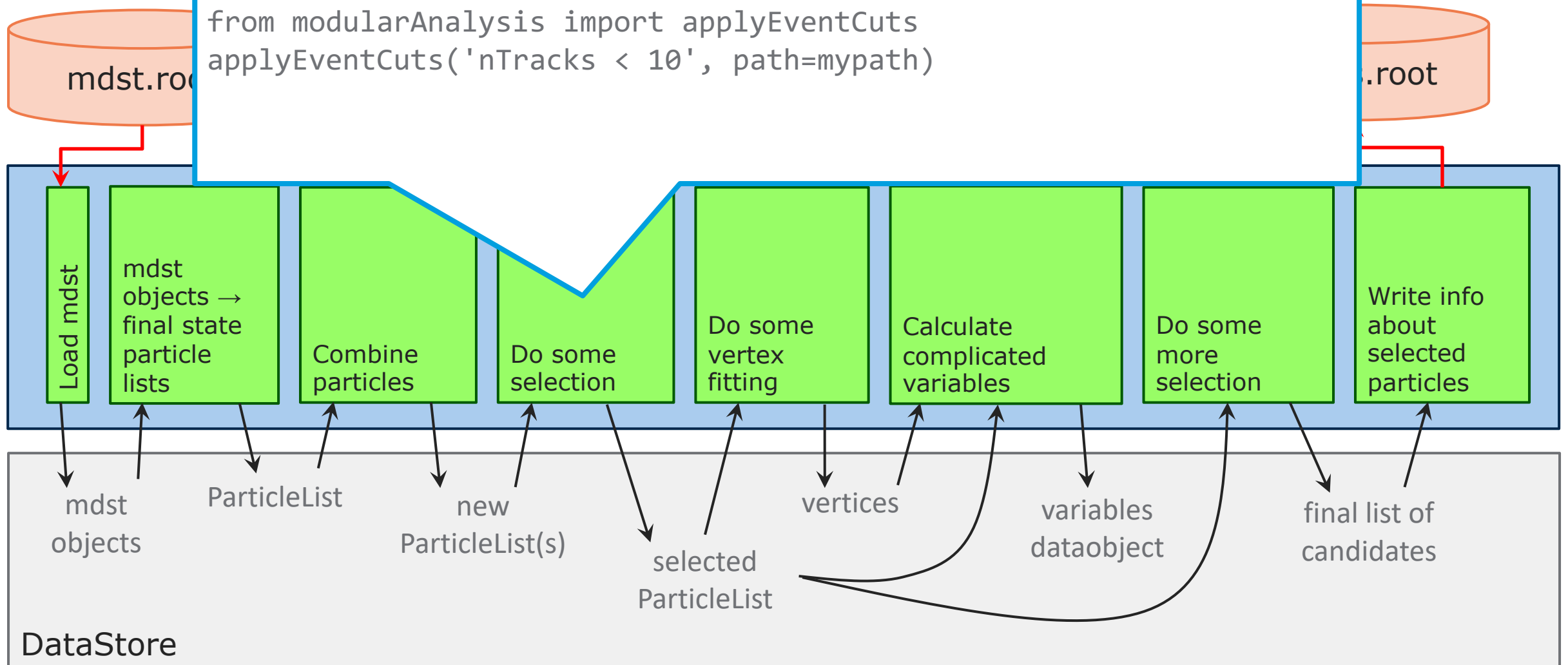
A

```
from modularAnalysis import reconstructDecay
reconstructDecay('K*0:myKst -> K+:highMom pi-:highMom', '0.6 < M < 1.0', path=mypath)
```

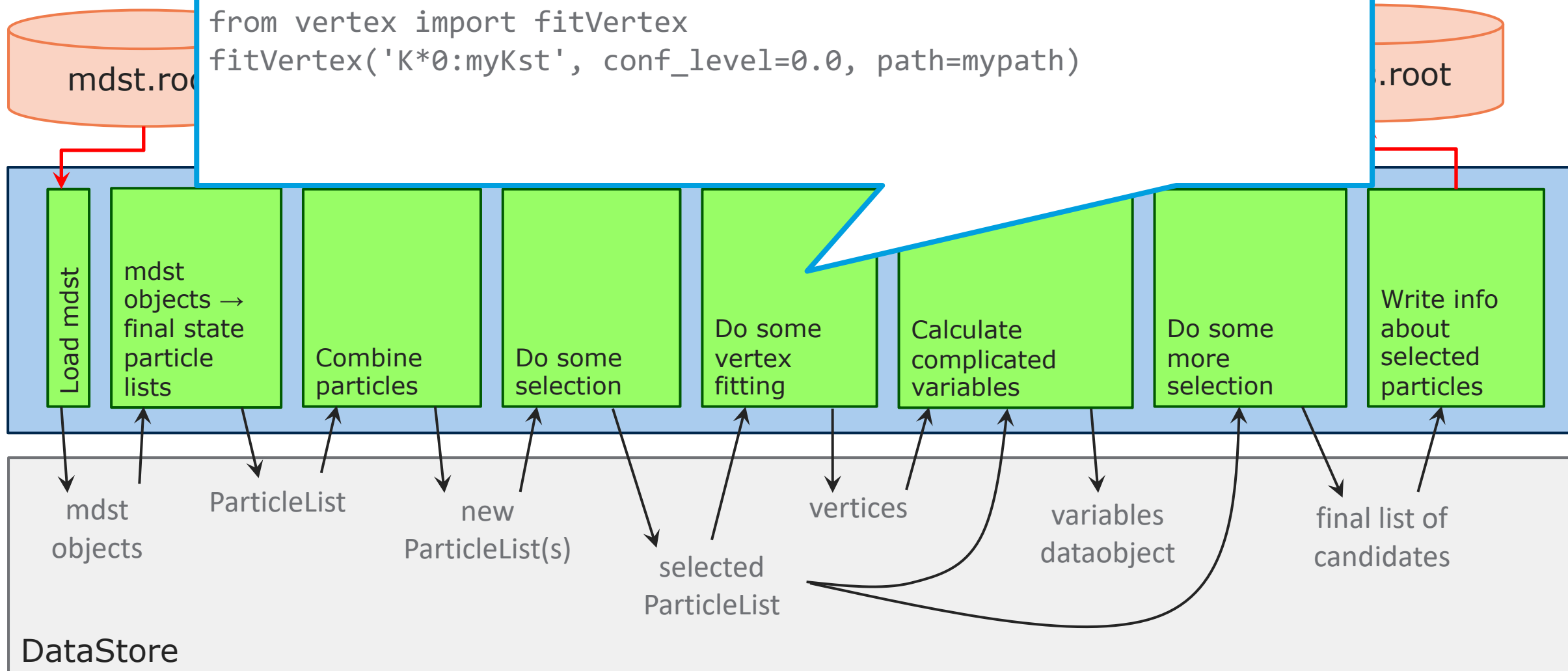
You can probably guess what this means already. But don't worry, we'll discuss the **DecayString** a bit later...



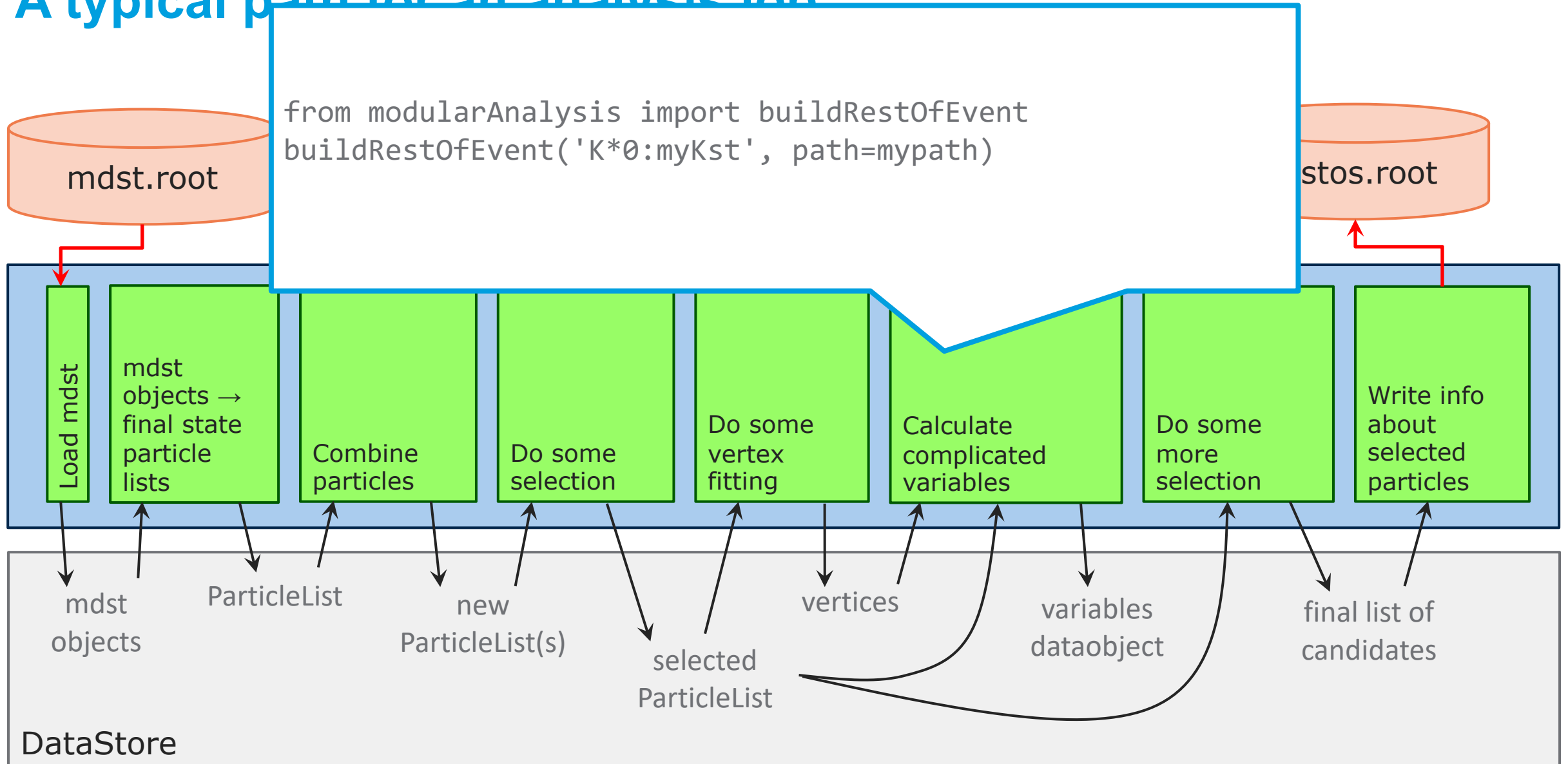
A typical event analysis pipeline



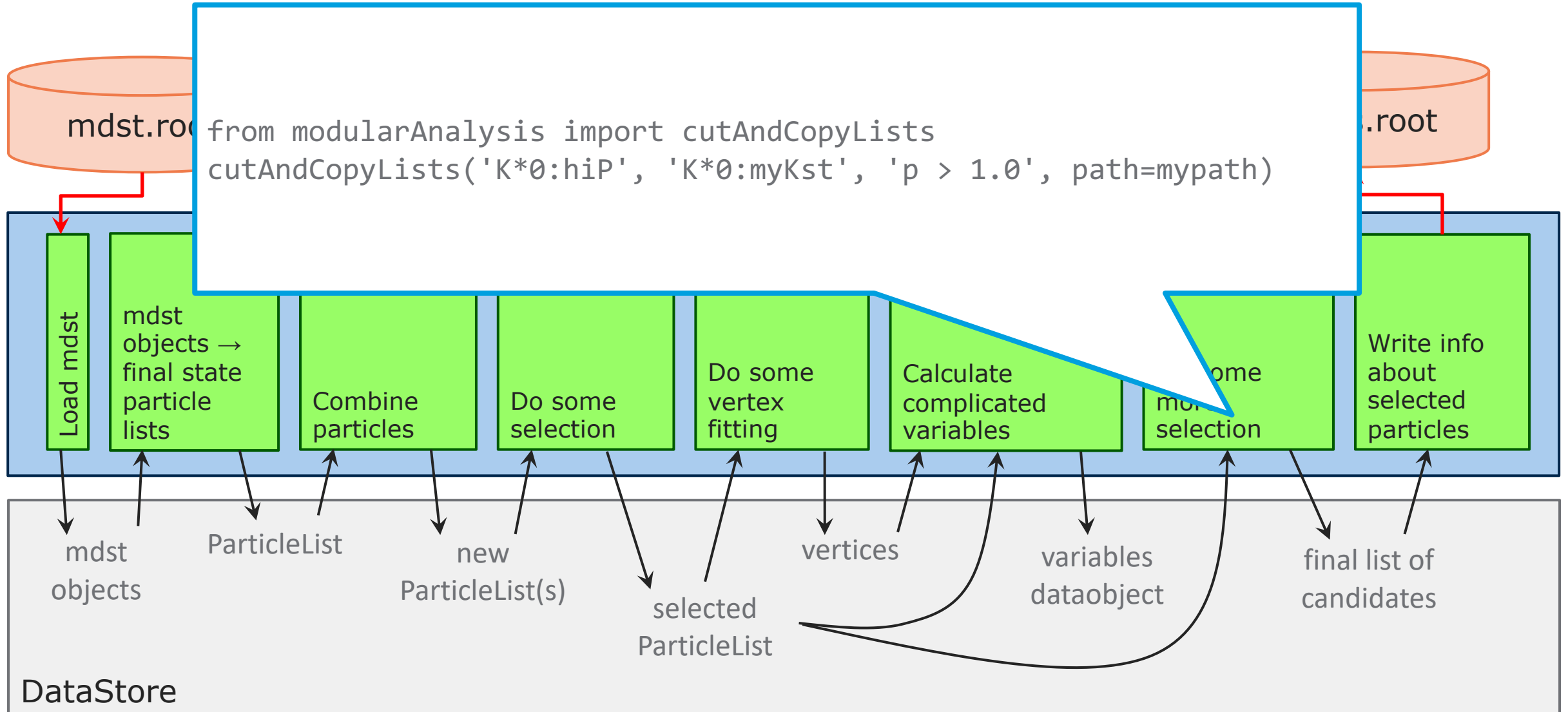
A typical workflow for manual analysis



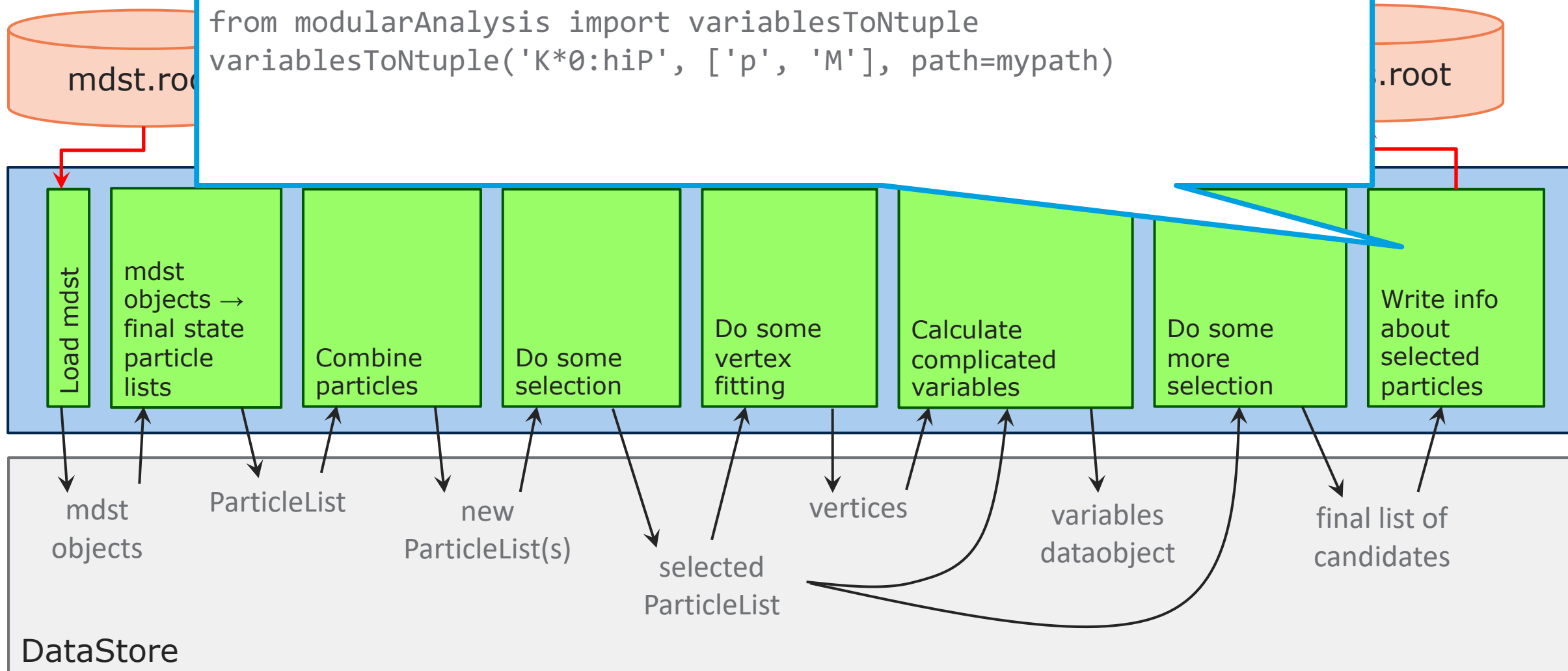
A typical path for an analysis job



A typical path for an analysis job



A typical event analysis pipeline



A template steering script

```
#!/usr/bin/env python3
# -*- coding: utf-8 -

import basf2
import modularAnalysis as ma

# create a path
mypath = basf2.Path()

# input mdst file
ma.inputMdst('default', 'inputMdstFile.root', path=mypath)

# PUT YOUR CODE HERE

# process the events
basf2.process(mypath)
# print out the summary
print(basf2.statistics)
```



A template steering script

```
#!/usr/bin/env python3
# -*- coding: utf-8 -

import basf2
import modularAnalysis as ma

# create a path
mypath = basf2.Path()

# input mdst file
ma.inputMdst('default', 'inputMdstFile.root', path=mypath)

# PUT YOUR CODE HERE

# process the events
basf2.process(mypath)
# print out the summary
print(basf2.statistics)
```

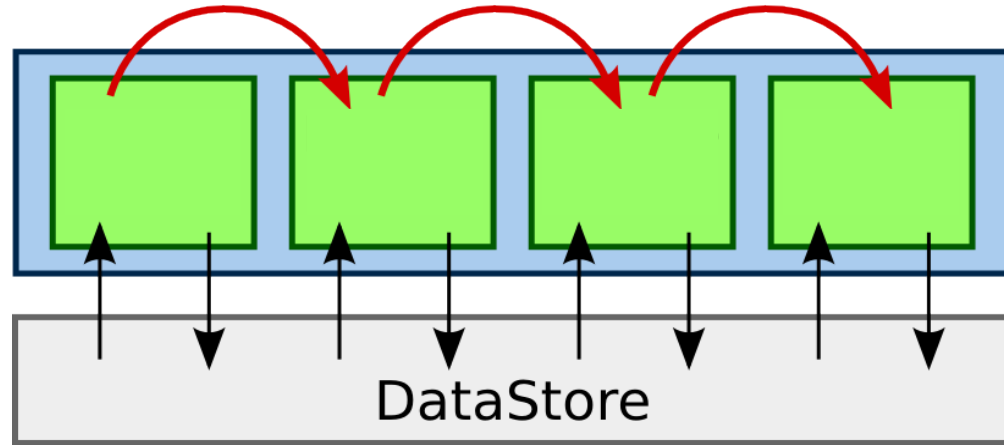


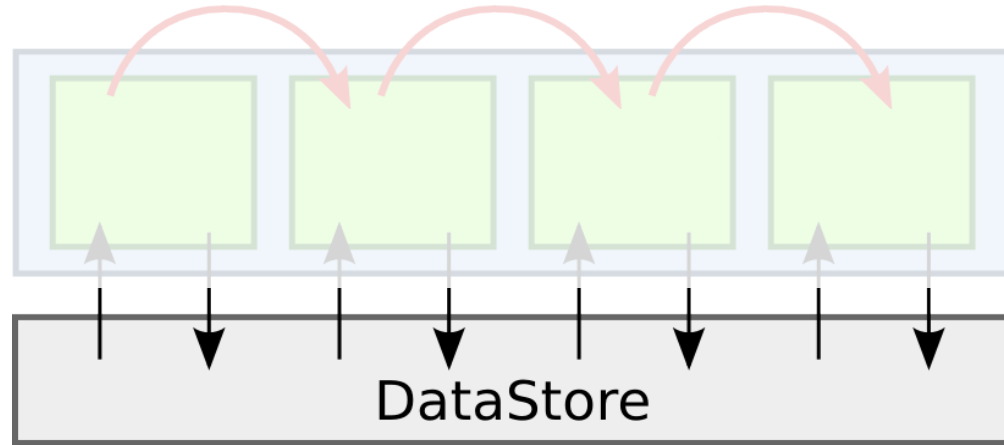
Q: what does 'default' mean?

What is the argument called?
Can you find the documentation?
Is there a questions post?

Candidate/particle based analysis

Before we get started, here's this diagram again





You can take a look at examples in `<package>/dataobjects`

- The datastore contains the **dataobjects**
- At the level of analysis, the main dataobject is: **ParticleList**

Particle-based paradigms

In the wild

**Event based analysis
frameworks**
Babar, ATLAS, ILC

**Particle/candidate
based analysis
frameworks**
LHCb, Belle II

Particle-based analysis

- Take particle lists
 - Build up decay parents from daughters
 - Make ***candidates*** for your decay of interest
 - Filter/cut/keep.
-
- You might have more than one candidate per event.
 - We deal with this after the fact.
 - This is fine. I promise.
 - <https://arxiv.org/abs/1703.01128>

The Particle class

It's not crucial to understand the details

- A common representation of all particle types
 - Charged: $e / \mu / \pi / K / p / d$ [built up from track + hypothesis]
 - γ [built up from ECLClusters + !Track]
 - K^0_L, n [built up from KLMLusters + ECLClusters + !Track]
 - K^0_S, Λ^0, γ [built up from V0 (2 tracks)]
 - Composite particles: $\pi^0 / K^0_S / D / B$ [built up from combinations]
- Data members of the class are **common to all particle types**: mass, momentum, position, PDG code, ...
- Information which is only relevant to certain kinds of particle is saved in separate **analysis package dataobjects** and accessible by **relations**.
 - e.g. ContinuumSuppression
 - FlavorTaggerInfo

The Particle class

It's not crucial to understand the details

- A common representation of all particle types
 - Charged: $e / \mu / \pi / K / p / d$ [built up from track + hypothesis]
 - γ [built up from ECLClusters + !Track]
 - K^0_L, n [built up from KLMLusters + ECLClusters + !Track]
 - K^0_S, Λ^0, γ [built up from V0 (2 tracks)]
 - Composite particles: $\pi^0 / K^0_S / D / B$ [built up from combinations]
- Data members of the class are **common to all particle types**: mass, momentum, position, PDG code, ...
- Information which is only relevant to certain kinds of particle is saved in separate **analysis package dataobjects** and accessible by **relations**.
 - e.g. ContinuumSuppression
 - FlavorTaggerInfo

Q: what is the name of the module to combine particles?

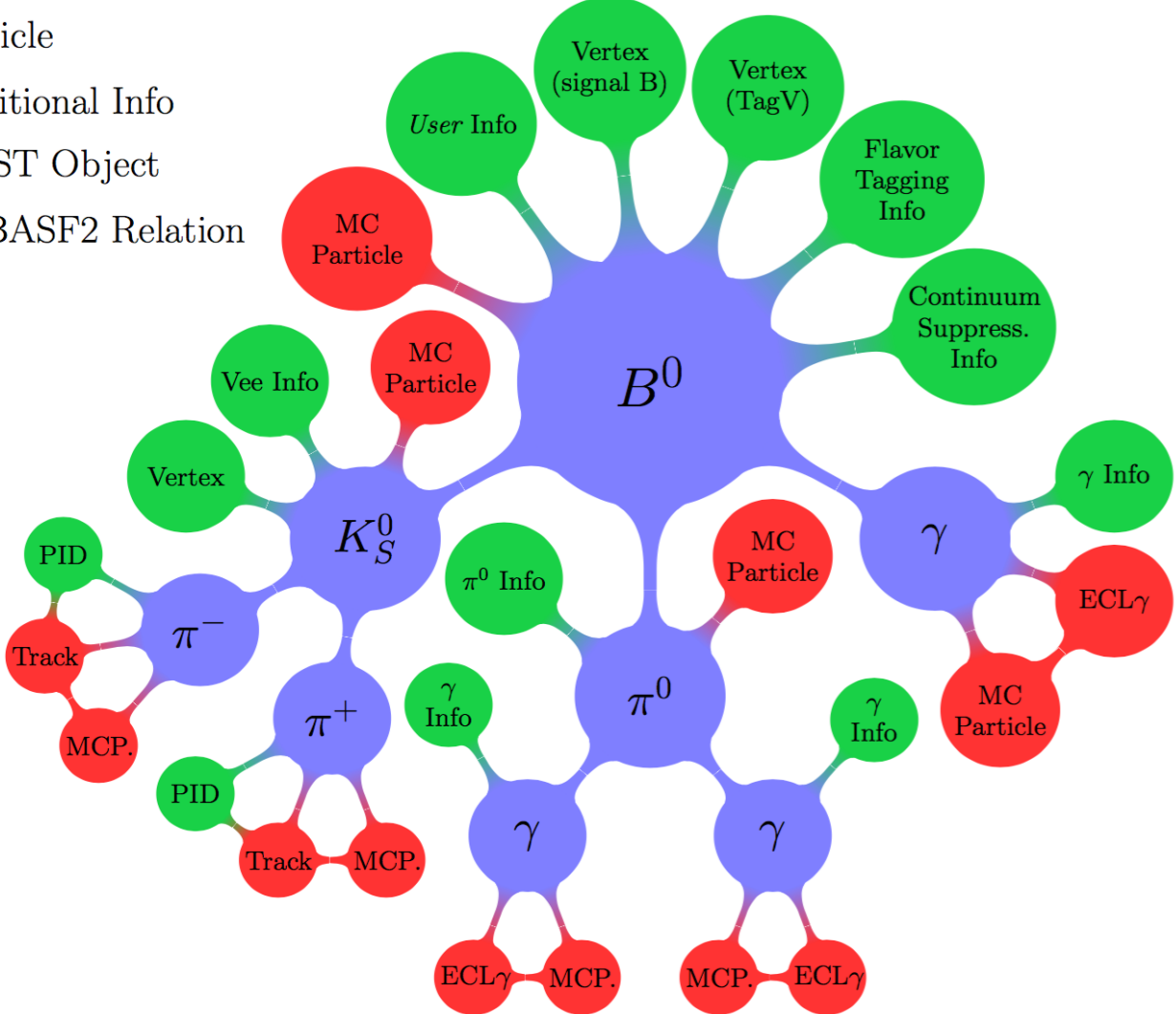
Q: which things on this slide are dataobjects?

ParticleList

- A group of all particles and anti-particles that belong together logically.
 - e.g. K^{*0} s (decaying to K^\pm and π^\mp with invariant mass in a certain window)
 - Can only store particles of the same PDG code (can be different decay modes).
 - Doesn't have ownership of the **Particle** objects.
 - **ParticleList** is the **dataobject** on which analysis modules operate.
-
- The physics-performance group provides **Standard Particle Lists** for which quality benchmarks exist, and systematics will be provided.
 - Currently you may need to optimize selection criteria of ParticleLists by yourself.
 - This will be really recommended to use in future.

Some more details on Particles, ParticleLists and Relations

- Particle
- Additional Info
- MDST Object
- BASF2 Relation



- At each stage we build relations between the dataobjects
- Like vertex information, ContinuumSuppression → all related to Particles
- Particles themselves related to primitive mdst objects (clusters, tracks)

Fill a ParticleList

There are two possible ways to fill a list of **stable*** particles

1) fill it by hand

```
import modularAnalysis as ma

ma.fillParticleList('pi+:highMom', 'p > 1', path=mypath)
```

Q: What is a stable particle in this context?

Fill a ParticleList

There are two possible ways to fill a list of **stable*** particles

1) fill it by hand

```
import modularAnalysis as ma  
  
ma.fillParticleList('pi+:highMom', 'p > 1', path=mypath)
```

Particle name.

Tells basf2 if the list has to be created from tracks, ELCClusters, KLMClusters or V0

Must given in evt.pdl (or b2help-particles)

```
evt.pdl = $BELLE2_EXTERNALS_DIR/share/evtgen/evt.pdl
```

Fill a ParticleList

There are two possible ways to fill a list of **stable*** particles

1) fill it by hand

```
import modularAnalysis as ma

ma.fillParticleList('pi+highMom', 'p > 1', path=mypath)
```

Particle name.

Tells basf2 if the list has to be created from tracks, ELCClusters, KLMClusters or V0

Must given in evt.pdl (or b2help-particles)

Label name.

Used to distinguish different lists of the same particle type: you choose the name.

(But maybe call it something helpful)

```
evt.pdl = $BELLE2_EXTERNALS_DIR/share/evtgen/evt.pdl
```


Fill a ParticleList

There are two possible ways to fill a list of **stable*** particles

1) fill it by hand

```
import modularAnalysis as ma  
  
ma.fillParticleList( pi+:highMom, 'p > 1', path=mypath)
```

Particle name.

Tells basf2 if the list has to be created from tracks, ELCClusters, KLMClusters or V0
Must given in evt.pdl (or b2help-particles)

Label name.

Used to distinguish different lists of the same particle type: you choose the name.
(But maybe call it something helpful)

ParticleList name

Same label name can be set to another particle name.
e.g. pi+:highMom, K+:highMom

```
evt.pdl = $BELLE2_EXTERNALS_DIR/share/evtgen/evt.pdl
```

Fill a ParticleList

There are two possible ways to fill a list of **stable*** particles

1) fill it by hand

```
import modularAnalysis as ma  
  
ma.fillParticleList('pi+:highMom', 'p > 1', path=mypath)
```

Particle name.

Tells basf2 if the list has to be created from tracks, ELCClusters, KLMClusters or V0

Must given in evt.pdl (or b2help-particles)

Label name.

Used to distinguish different lists of the same particle type: you choose the name.

(But maybe call it something helpful)

Selection criteria

Cuts used to select this list. **More on this later**

List name

Same label name can be set to another particle name.

e.g. pi+:highMom, K+:highMom

evt.pdl = \$BELLE2_EXTERNALS_DIR/share/evtgen/evt.pdl

Fill a ParticleList

There are two possible ways to fill a list of **stable* particles**

1) fill it by hand

```
import modularAnalysis as ma  
  
ma.fillParticleList('pi+:highMom', 'p > 1', path=mypath)
```

Once you call the function, pi-:highMom is created automatically.
You **don't** have to call

```
ma.fillParticleList('pi-:highMom', 'p > 1', path=mypath)
```

```
evt.pdl = $BELLE2_EXTERNALS_DIR/share/evtgen/evt.pdl
```

Fill a ParticleList

There are two possible ways to fill a list of **stable*** particles

1) fill it by hand

```
import modularAnalysis as ma

ma.fillParticleList('pi+:highMom', 'p > 1', path=mypath)
```

2) Use the **standard particles lists** with pre-defined names and cuts

```
from stdPhotons import stdPhotons
from stdCharged import stdK

stdPhotons('all', path=mypath) # --> gamma:all
stdK('95eff', path=mypath)     # --> K+:95eff
```

First argument will be a label name of the particle list.

```
evt.pdl = $BELLE2_EXTERNALS_DIR/share/evtgen/evt.pdl
```

DecayString

The last of the infrastructure things, I promise

- You've already seen the decay string in action.
- Now we can formally describe what this line is doing:

```
reconstructDecay('K*0:myKst -> K+:highMom pi-:highMom', '0.6 < M < 1.0', path=mypath)
```

DecayString

The last of the infrastructure things, I promise

- You've already seen the decay string in action.
- Now we can formally describe what this line is doing:

```
reconstructDecay('K*0:myKst -> K+:highMom pi-:highMom' '0.6 < M < 1.0', path=mypath)
```

Decay string

DecayString

The last of the infrastructure things, I promise

- You've already seen the decay string in action.
- Now we can formally describe what this line is doing:

```
reconstructDecay(K*0:myKst -> K+:highMom pi-:highMom '0.6 < M < 1.0', path=mypath)
```

ParticleList ParticleList ParticleList

Decay string

DecayString

The last of the infrastructure things, I promise

- You've already seen the decay string in action.
- Now we can formally describe what this line is doing:

```
reconstructDecay('K*0:myKst -> K+:highMom pi-:highMom' '0.6 < M < 1.0', path=mypath)
```

Arrow
(Indicate K*0 decays to K+ pi-)

Decay string

DecayString

The last of the infrastructure things, I promise

- You've already seen the decay string in action.
- Now we can formally describe what this line is doing:

```
reconstructDecay('K*0:myKst -> K+:highMom pi-:highMom '0.6 < M < 1.0', path=mypath)
```

```
mother arrow daughter0 daughter1 ...
```

Decay string

DecayString

The last of the infrastructure things, I promise

- You've already seen the decay string in action.
- Now we can formally describe what this line is doing:

```
reconstructDecay(K*0:myKst -> K+:highMom pi-:highMom' '0.6 < M < 1.0', path=mypath)
```

Particle name.
Of your composite particle.
Must given in evt.pdl
(or b2help-particles)

Decay string

```
evt.pdl = $BELLE2_EXTERNALS_DIR/share/evtgen/evt.pdl
```

DecayString

The last of the infrastructure things, I promise

- You've already seen the decay string in action.
- Now we can formally describe what this line is doing:

```
reconstructDecay('K*0:myKst -> K+:highMom pi-:highMom' '0.6 < M < 1.0', path=mypath)
```

You can choose this.

List name that exist already

Decay string

- `reconstructDecay()` : Reconstruct mother particle from daughters with a given DecayString. List names for daughters must exist already. You can choose a list name of a mother.

You don't have to call

```
reconstructDecay('anti-K*0:myKst -> K-:95eff pi+:95eff', '0.6 < M < 1.0', path=mypath)
```

DecayString

One final thing...

- Sometimes you will need to select a particle within a decay string.
- This is done by "carat": ^

```
'K*0:myKst -> ^K+:highMom ^pi-:highMom'
```

- You will see this in action in the exercises later.

Physics quantities and the VariableManager

The VariableManager

It manages variables

- VariableManager is a place in the analysis package to store variables
 - physics quantities: invariant mass, beam-constrained mass, E , p , p_T , θ , ϕ , highest energy in a cluster
 - counters: event_number, nhits, i_candidate
- Every variable takes at least a Particle* as input and returns a double (even integer counters like event_number where this doesn't make much sense)



The VariableManager

It manages variables

- VariableManager is a place in the analysis package to store variables
 - physics quantities: invariant mass, beam-constrained mass, E , p , p_T , θ , ϕ , highest energy in a cluster
 - counters: event_number, nhits, i_candidate
- Every variable takes at least a Particle* as input and returns a double (even integer counters like event_number where this doesn't make much sense)
- You've already seen it in **python**, when we used a cut on "M".
 - This comes from the VariableManager.

```
reconstructDecay('K*0:myKst -> K+:highMom pi-:highMom', '0.6 < M < 1.0', path=mypath)
```



Where can I get the documentation?

- At the command line:

```
$ basf2 variables.py
```

- Online, the documentation is software.belle2.org.

Take a look at B2T_Basics_1_GettingHelp.ipynb on jupyterhub

Q: What is the beam-constrained mass, M_{bc} called in the VariableManager?

Have you been listening?

Q: Can you figure out the cut string and code to reconstruct B0 -> K*0 gamma?
Supposing you have already reconstructed K*0:myKst.

```
import modularAnalysis as ma

ma.fillParticleList('pi+:highMom', 'p > 1', path=mypath)
ma.fillParticleList('K+:highMom', 'p > 1', path=mypath)
ma.reconstructDecay('K*0:myKst -> K+:highMom pi-:highMom', '0.6 < M < 1.0', path=mypath)
```

Q: How can we fill a particle list of gamma?

Q: How can we reconstruct B0 from K*0 and gamma?

Q: Can you apply following selection on the B0 candidates?

$$M_{bc} > 5.2 \text{ GeV}/c^2$$

Aliases

Are awesome and you should use them

- With more advanced variables, we deliberately give them verbose names in the VariableManager.
- It's important to be clear.
- Some examples:
 - `cosAngleBetweenMomentumAndVertexVector`
 - `totalPhotonEnergyOfEvent`
 - `trackFindingFailureFlag`
- You can also use `formula()`, `abs()`, `cos()` for simple math:
 - `formula(missingEnergyOfEventCMS/Ecms)`
 - ...

Aliases

Are awesome and you should use them

- For your code, you can make short aliases.
- This makes your offline data more manageable, and can help with code readability.

```
from variables import variables as vm
vm.addAlias('cosPVtx',
           'cosAngleBetweenMomentumAndVertexVector')
```

There is no path here like path=myspath.
The VariableManager exists alongside the path

How to get data out

Flat ntuples and histograms

Getting output data

There are some modules to store variables.

VariablesTo*

- a) VariablesToNtuple
- b) VariablesToHistogram
- c) VariablesToEventBasedTree

Flat ntuples and histograms

Getting output data

There are some modules to store variables.

VariablesTo*

- a) VariablesToNtuple
- b) VariablesToHistogram
- c) VariablesToEventBasedTree

```
from modularAnalysis import variablesToNtuple  
from modularAnalysis import variablesToHistogram
```

VariablesToNtuple

- You want to store physics quantities for one ParticleList from the VariableManager.
- Get an ttree (ntuple) of candidates (one row per candidate).

VariablesToHistogram

- Perhaps you just want a quick histogram.
- Candidate information not preserved.

VariablesToNtuple

```
# ... at the end of your script
from modularAnalysis import variablesToNtuple
variablesToNtuple('K*0:myKst', ['M', 'p', 'daughter(0,p)', 'daughter(1,p)'], path=mypath)

# process the events
basf2.process(mypath)
```

Daughter's variables can be stored with `daughter(i, variable)`.
`i` is the daughter index, i.e. `0 = K+`, `1 = pi-`.

VariablesToHistogram

```
# ... at the end of your script
from modularAnalysis import variablesToHistogram
variablesToHistogram('K*0:myKst',
                    [ ('M', 100, 0.7, 1.0),
                      ('p', 100, 0.1, 3.0) ], path=mypath)

# process the events
basf2.process(mypath)
```


VariablesToNtuple

```
# ... at the end of your script
from modularAnalysis import variablesToNtuple
variablesToNtuple('K*0:myKst', ['M', 'p', 'daughter(0,p)', 'daughter(1,p)'], path=mypath)

# process the events
basf2.process(mypath)
```

Daughter's variables can be stored with `daughter(i, variable)`.
`i` is the daughter index, i.e. `0 = K+`, `1 = pi-`.

VariablesToHistogram

```
# ... at the end of your script
from modularAnalysis import variablesToHistogram
variablesToHistogram('K*0:myKst',
                    [ ('M', 100, 0.7, 1.0),
                      ('p', 100, 0.1, 3.0) ], path=mypath)

# process the events
basf2.process(mypath)
```

Q: Can you guess what these numbers do?
What is the (python) structure created by the parentheses?

Utilities

Helper functions to make daughter's aliases.

- You can store variables of daughters with `daughter(i, variable)` in a same tuple.
- You may also want to short aliases for these variables.

```
reconstructDecay('K*0:myKst -> K+:highMom pi-:highMom', '0.6 < M < 1.0', path=mypath)

from variables import variables as vm
vm.addAlias('K_M', 'daughter(0,M)')
vm.addAlias('K_p', 'daughter(0,p)')
vm.addAlias('pi_M', 'daughter(1,M)')
vm.addAlias('pi_p', 'daughter(1,p)') # You may want to add more aliases

vars = ['M', 'p', 'K_M', 'K_p', 'pi_M', 'pi_p']
variablesToNtuple('K*0:myKst', vars, path=mypath)
```

Utilities

Helper functions to make daughter's aliases.

- A helper function creates aliases from a list of variables and a DecayString with carat '^'.

```
import variables.utils as vu
vars = vu.create_aliases_for_selected(['M', 'p', 'px', 'py', 'pz'],
                                     'K*0 -> ^K+ ^pi-')
# The function makes aliases for selected particles, for example,
# 'K_M' --> 'daughter(0,M)'
```

- All aliases can be checked with printAliases().

```
from variables import variables as vm
vm.printAliases()
# [INFO] =====
# [INFO] Following aliases exists:
# [INFO] 'K_M' --> 'daughter(0,M)'
# ...
```

Nomenclature

Nomenclature

<https://confluence.desy.de/display/BI/Main+Glossary>

- **Experiment** (chunk of data-taking ~months).
- **Run** (chunk of data-taking w/ stable beams ~hours),
- Event.

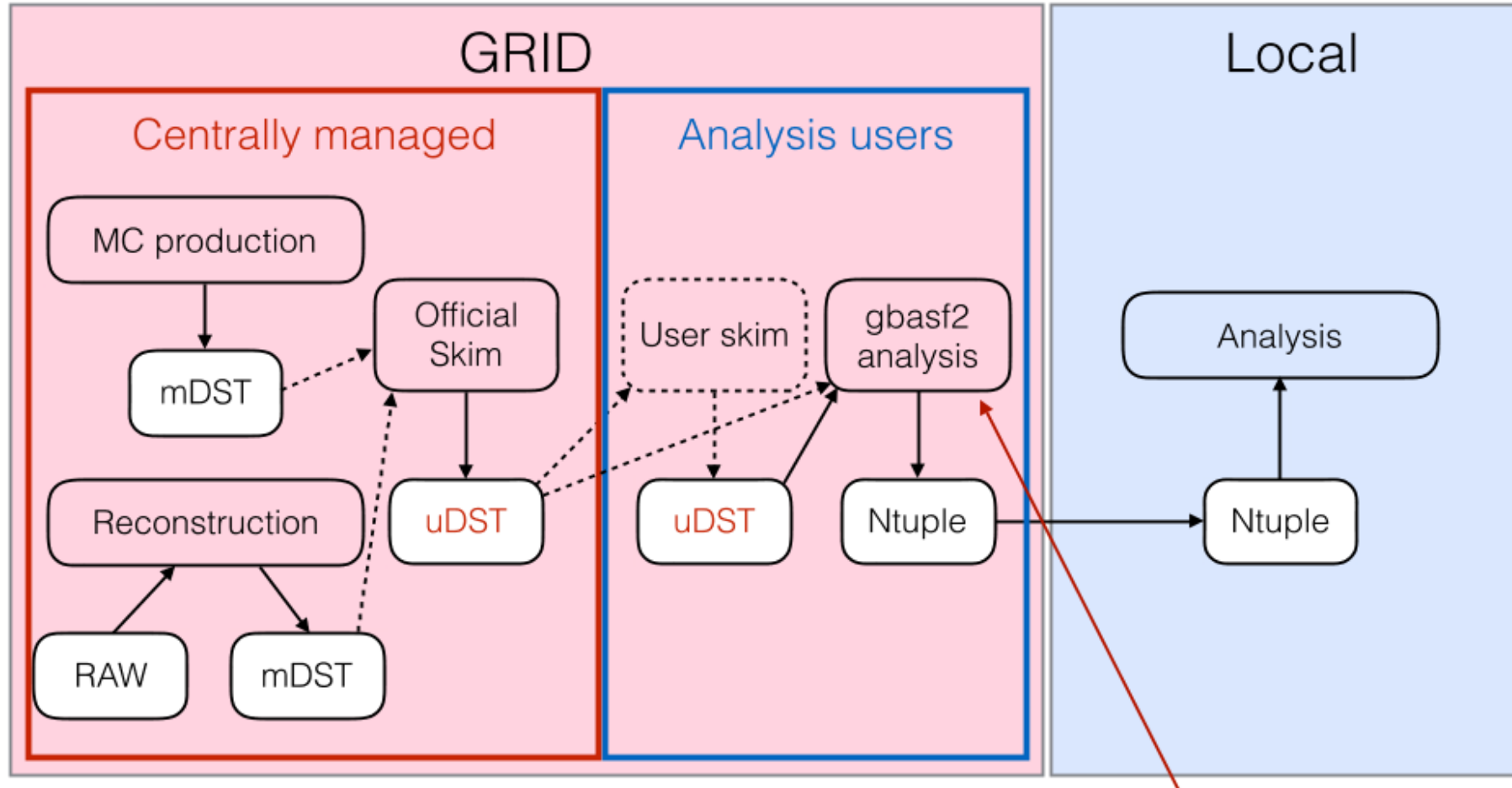
- **TRG** the hardware trigger (group, device, DAQ)
- **L1** the hardware trigger (used interchangeably)

- SoftwareTrigger / **HLT** (the software trigger)

- **basf2** “Belle 2 analysis software framework” “the software”
- **gbasf2** “The grid job submission tool” “computing”

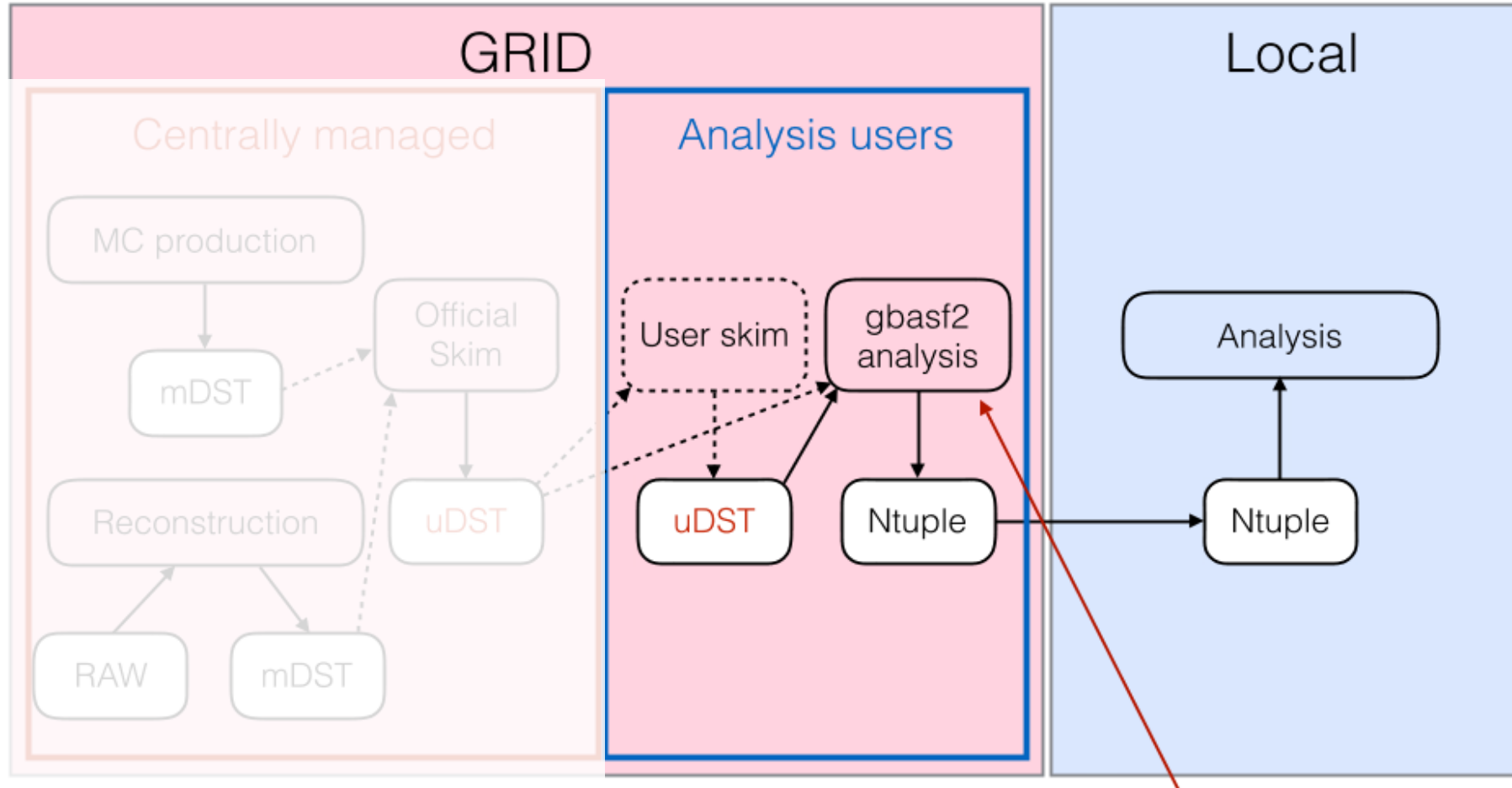
One final thing about Belle vs. Belle II

The Belle II analysis model



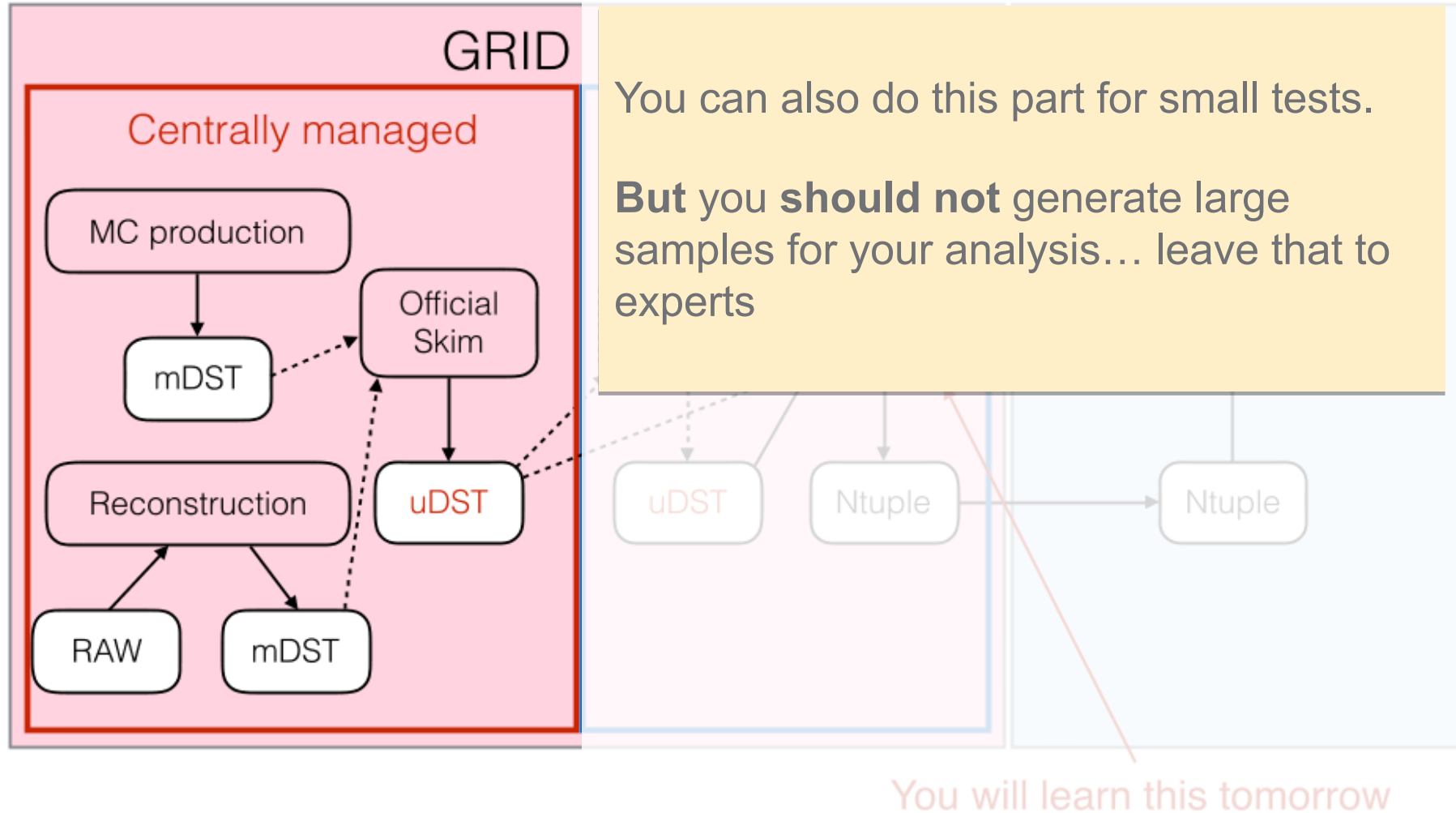
You will learn this tomorrow

The Belle II analysis model



You will learn this tomorrow

The Belle II analysis model



Acknowledgements

- Much of the material for these slides (and ideas for how to present it) has been stolen from:

Jake Bennett and Anže Zupanc

Versions of this tutorial material

- Version 1: Sam (February 2018).
- Version 2: Ilya and Sam (June 2018).
- Version 3: Sam (November 2018).
- Version 4: Umberto (February 2019)
- Version 5: Hannah and Sam (June 2019)
- Version 6: Yo and Ilya (January 2020)